Software Testing Basics

软件测试基础

周元哲 主编

周元哲 胡滨 潘晓英 刘海 编著

西安电子科技大学出版社
http://www.xduph.com

XDUP 278300
封面设计：Wisdom 汇智策劃

读者对象

高等院校相关专业师生，从事计算机应用开发的各类技术人员，参加全国计算机软件测评、软件技术资格与水平考试人员。

Software Testing Basics

ISBN 978-7-5606-2491-4
9 787560 624914 >
定价：30.00元

软件测试基础

周元哲　主编

周元哲　胡　滨　潘晓英　刘　海　编著

西安电子科技大学出版社

内 容 简 介

本书较为全面、系统地介绍了当前业界测试领域的理论和实践知识，反映了当前最新的软件测试理论、标准、技术和工具，展望了软件测试的发展趋势。

全书共分三大部分，分别是测试理论、测试实践和测试考试指导。第一部分内容主要包括软件测试概论、软件测试基本知识、软件测试过程、黑盒测试、白盒测试、自动测试技术、性能测试、面向对象测试、嵌入式测试和软件测试管理。第二部分内容主要包括软件测试工具、测试管理工具、性能测试工具、缺陷跟踪管理工具、单元测试工具和功能测试工具等。第三部分内容主要包括计算机认证考试和测试行业，介绍了四级软件测试工程师考试和企业招聘测试工程师考试的一些情况。

本书可作为高等院校相关专业软件测试课程的教材或教学参考书，也可供从事计算机应用开发的各类技术人员参考，或用作全国计算机软件测评师考试、软件技术资格与水平考试的培训资料。

图书在版编目(CIP)数据

软件测试基础 / 周元哲主编. —西安：西安电子科技大学出版社，2011.6

ISBN 978-7-5606-2491-4

Ⅰ. ① 软… Ⅱ. ① 周… Ⅲ. ① 软件—测试 Ⅳ. ① TP311.5

中国版本图书馆 CIP 数据核字(2010)第 202970 号

策　　划　云立实

责任编辑　杨丕勇　云立实

出版发行　西安电子科技大学出版社(西安市太白南路 2 号)

电　　话　(029)88242885　88201467　　邮　　编　710071

网　　址　www.xduph.com　　电子邮箱　xdupfxb001@163.com

经　　销　新华书店

印刷单位　陕西天意印务有限责任公司

版　　次　2011 年 6 月第 1 版　2011 年 6 月第 1 次印刷

开　　本　787 毫米×1092 毫米　1/16　印　张　16

字　　数　377 千字

印　　数　1～3000 册

定　　价　30.00 元

ISBN 978-7-5606-2491-4/TP・1241

XDUP 2783001-1

前　言

随着软件规模和复杂性的大幅度提升，软件质量可靠性的问题变得日益突出。软件测试是保证软件质量的关键技术之一，同时也是软件开发过程中的一个重要环节，其理论知识和技术工具都在不断革新。

本书较为全面地介绍了当前业界测试领域的专业知识，追溯了软件测试的发展史，反映了当前最新的软件测试理论、标准、技术和工具，展望了软件测试的发展趋势。全书共分三大部分，分别是测试理论、测试实践和测试考试指导。第一部分内容主要包括软件测试概论、软件测试基本知识、软件测试过程、黑盒测试、白盒测试、自动测试技术、性能测试、面向对象测试、嵌入式测试和软件测试管理。第二部分内容主要包括软件测试工具、测试管理工具、性能测试工具、缺陷跟踪管理工具、单元测试工具和功能测试工具等。第三部分内容主要包括计算机认证考试和测试行业，介绍了四级软件测试工程师考试和企业招聘测试工程师考试的一些情况。

软件测试从软件工程中演化而来，并且还在不断地发展。在学习本书之前，需要一些先行知识作为本书的支撑，如掌握一门高级语言(Visual Basic 或 Java 语言)、数据库、数据结构以及软件工程的基本理论知识等。

本书介绍了软件测试的基本理论和当前流行的一些软件测试工具的应用，内容精练，文字简洁，结构合理，综合性强，定位明确，面向初、中级读者，由“入门”起步，侧重“提高”，特别适合作为高等院校相关专业软件测试的教材或教学参考书，也可供从事计算机应用开发的各类技术人员参考，或用作全国计算机软件测评师考试、软件技术资格与水平考试的培训资料。

本书由周元哲、胡滨、潘晓英、刘海编写，其中第 13、16 章由胡滨编写，第 9、12 章由潘晓英编写，第 14 章由刘海编写，其余章节由周元哲编写。周元哲任主编，负责拟订大纲与统稿工作。

西安邮电学院计算机科学与技术学院王忠民院长、王曙燕副院长和陈莉君教授对本书的编写给予了大力的支持并提出了指导性意见。

在本书的写作过程中，西安电子科技大学出版社的云立实、人民邮电出版社的贾楠对写作大纲、写作风格等提出了很多宝贵的意见。本书在写作过程中参阅了大量中外文的专著、教材、论文、报告及网上的资料，由于篇幅所限，未能一一列出。在此，向各位作者表示敬意和衷心的感谢。

由于作者水平有限，本书难免有不足之处，诚恳期待读者的批评指正，以使本书日臻完善。我的电子信箱是 zhouyuanzhe@163.com。

编　者

2010 年 10 月

目　录

第1章 软件测试概论

本章主要介绍软件的发展历史，当前流行的软件过程模型、软件缺陷、软件质量以及质量保证体系等理论知识，引出软件测试对于软件开发和软件质量的重要性，为学习本书后续内容作必要准备。

1.1 软 件

软件是一系列按照特定顺序组织的计算机数据和指令的集合。一般来讲，软件被划分为编程语言、系统软件、应用软件和介于系统软件与应用软件之间的中间件。其中系统软件为使用计算机提供最基本的功能，但是并不针对某一特定应用领域；而应用软件则恰好相反，不同的应用软件根据用户和所服务的领域提供不同的功能。

一般认为，软件包括如下内容：

(1) 运行时，能够提供所要求功能和性能的指令或计算机程序集合。

(2) 程序能够妥善地处理信息的数据结构。

(3) 描述程序功能需求以及程序如何操作和使用的文档。

1.1.1 软件发展史

软件的发展经历了如下几个阶段：

第一阶段从20世纪50年代初期至60年代中期，这一阶段又称为程序设计阶段。此时硬件已经通用化，而软件的生产却是个体化。软件产品为专用软件，规模较小，功能单一，开发者即为使用者，软件只有程序，无文档。软件设计在人们的头脑中完成，形成了“软件等于程序”的错误观念。

第二阶段从20世纪60年代中期至70年代末期，称为程序系统阶段。此时多道程序设计技术、多用户系统、人机交互技术、实时系统和第一代数据库管理系统的出现，催生了专门从事软件开发的“软件作坊”，软件广泛应用，但软件技术和管理水平相对落后，导致“软件危机”出现。软件危机主要表现在以下几个方面：

(1) 软件项目无法按期完成，超出经费预算，软件质量难以控制；

(2) 开发人员和开发过程之间管理不规范，约定不严密，文档书写不完整，使得软件维护费用高，某些系统甚至无法进行修改；

(3) 缺乏严密、有效的质量检测手段，交付给用户的软件质量差，在运行中出现许多问题，甚至产生严重的后果；

(4) 系统更新换代难度大。

第三阶段从20世纪70年代末期至80年代中期称为软件工程阶段。微处理器的出现及分布式系统的广泛应用，使得计算机真正成为大众化的东西。以软件的产品化、系列化、工程化和标准化为特征的软件产业发展起来，软件开发有了可以遵循的软件工程化的设计准则、方法和标准。1968年，北大西洋公约组织的计算机科学家在联邦德国召开国际会议，会议讨论了软件危机问题，正式提出并使用“软件工程”概念。这标志着软件工程的诞生。

软件工程学涉及与生产软件相关的所有活动，相关的学科包括计算机科学、管理学、经济学、心理学等。软件工程研究的主要内容是：如何应用科学的理论和工程上的技术来指导软件的开发，从而达到以较少的投资获得高质量软件的最终目标。

第四阶段从20世纪80年代中期至今，客户端/服务器(C/S)体系结构，特别是Web技术和网络分布式对象技术的飞速发展，导致软件系统体系结构向更加灵活的多层分布式结构演变，CORBA、EJB、COM/DCOM等三大分布式的对象模型技术相继出现。

2006年提出的面向服务架构(Service-Oriented Architecture，简称SOA)作为下一代软件架构，是一种“抽象、松散耦合和粗粒度”的软件架构，根据需求通过网络对松散耦合的粗粒度应用组件进行分布式部署、组合和使用，主要用于解决传统对象模型中无法解决的异构和耦合问题。

至此，软件发展经历了从Mainframe结构、Client/Server结构、B/S多层分布式结构到SOA的演变过程，整个软件系统变得越来越分散、越来越开放、越来越强调互操作性。

1.1.2 软件生命周期

同任何事物一样，一个软件产品或软件系统也要经历孕育、诞生、成长、成熟、衰亡等阶段，一般称为软件生命周期。通过将整个软件生命周期划分为若干阶段，可使得每个阶段有明确的任务，使规模大且结构和管理复杂的软件开发变得容易控制和管理。通常，软件生命周期包括可行性分析与开发项目计划、需求分析、设计(概要设计和详细设计)、编码、测试、维护升级直至废弃等阶段，这种按时间划分过程的思想方法是软件工程中的一种思想原则，即按部就班、逐步推进，每个阶段都要有定义、工作、审查并形成文档，以提高软件的质量。

软件生命周期具有如下六个阶段：

(1) 问题的定义及规划。此阶段由软件开发方与需求方共同讨论，确定软件的开发目标及其可行性。

(2) 需求分析。在确定软件开发可行的情况下，对软件需要实现的各个功能进行详细分析。需求分析阶段作为一个很重要的阶段，在整个软件开发过程中是不断变化和深入的，因此必须制定需求变更计划来应付这种变化，以保护整个项目的顺利进行。

(3) 软件设计。此阶段主要根据需求分析的结果，对整个软件系统进行设计，如系统框架设计、数据库设计等。软件设计一般分为总体设计和详细设计。

(4) 程序编码。此阶段是将软件设计的结果转换成计算机可运行的程序代码。程序编码必须符合标准的编写规范，以保证程序的可读性、易维护性，提高程序的运行效率。

(5) 软件测试。在软件设计完成后要经过严密的测试，以发现软件在整个设计过程中存在的问题并加以纠正。整个测试过程分为单元测试、组装测试以及系统测试等

阶段。测试的方法主要有白盒测试和黑盒测试两种。在测试过程中需要建立详细的测试计划并严格按照测试计划进行测试，以减少测试的随意性。

(6) 运行维护。软件维护是软件生命周期中持续时间最长的阶段。在软件开发完成并投入使用后，由于多方面的原因，软件有可能不适应用户的要求，要延续软件的使用寿命，此时就必须对软件进行维护。

1.1.3 软件缺陷

1. 软件缺陷案例

让我们回顾一些“臭名昭著”的软件缺陷案例，它们都是由于软件测试不充分而导致的严重问题。

1963 年，由于用 FORTRAN 程序设计语言编写的飞行控制软件中的循环语句“DO 5 I=1，3”误写为“DO 5 I=1.3”，结果导致美国首次金星探测飞行失败，造成价值约 1000 多万美元的损失。

1979 年，新西兰航空公司的一架客机因计算机控制的自动飞行系统发生故障而撞在阿尔卑斯山上，机上 257 名乘客全部遇难。

1983 年，美国科罗拉多河水泛滥，但由于计算机对天气形势预测有误，水库未能及时泄洪，以致造成严重的经济损失和人员伤亡。

1990 年 1 月 15 日，通信中转系统软件发生故障，导致主干远程网大规模崩溃，使数以千计的电信运营公司损失惨重。

1992 年 10 月 26 日，伦敦救护中心的计算机辅助发送系统刚启动就崩溃了，导致这个全世界最大的(每天要接运五千多名病人)救护机构全部瘫痪。

1994 年，美国迪士尼公司的“狮子王”软件在少数系统中能正常工作，但在大众使用的常见系统中无法正常运行。后来证实，这是由于迪士尼公司没有对市场上投入使用的各种 PC 机型进行正确的测试。同年，英特尔奔腾浮点除法发生软件缺陷，英特尔为处理软件缺陷支付了 4 亿多美元。

1996 年 6 月 4 日，欧洲航空航天局耗资 80 亿美元发射的阿里亚娜 501 火箭，在发射升空 37 秒后爆炸。原因是主发动机打火顺序开始 37 秒后，制导信息由于惯性制导系统的软件出现规格和设计错误而完全丢失。

临近 2000 年时，计算机业界一片恐慌，导致这种现象发生的就是著名的“千年虫”问题。其原因是在 20 世纪 70 年代，由于计算机硬件资源很珍贵，程序员为节约内存资源和硬盘空间，在存储日期数据时，只保留年份的后 2 位，如“1980”被存储为“80”。当 2000 年到来时，问题出现了，计算机无法分清“00”是指“2000 年”还是“1000 年”。例如银行存款的软件在计算利息时，用日期“00”减去当时存款的日期，结果存款年数就变为负数，导致顾客反要付给银行巨额的利息。为了解决“千年虫”问题，花费了大量的人力、物力和财力。

2008 年，我国举办了首次奥运会。2007 年 10 月 30 日上午 9 时，北京奥运会门票面向境内公众销售第二阶段正式启动，系统访问流量猛增，官方票务网站流量瞬时达到每小时 800 万次，超过了系统设计每小时 100 万次的承受量，奥运门票系统访问量超过原计划的 8

倍，造成网络拥堵，售票速度慢或暂时不能登录系统的情况，直接造成公众无法及时提交购票申请，北京奥运票务中心就此向广大境内公众购票人发布了致歉信。

2. 软件缺陷概念

软件缺陷是指软件系统或系统部件中那些导致系统或部件不能实现其功能的问题或差错。通常认为，符合下面 4 个规则之一的就是软件缺陷。

(1) 软件未达到软件规格说明书中规定的功能；

(2) 软件出现了产品说明书中指明不会出现的错误；

(3) 软件功能超出了产品说明书中指明的范围；

(4) 软件测试人员认为软件难于理解，不易使用，运行速度慢，或者最终用户认为软件使用效果不好。

【例 1-1】 软件缺陷举例。

计算器说明书一般声称该计算器能准确无误地进行加、减、乘、除运算。如果测试人员选定了数值，按下“+”号后，再按下一数值，没有任何结果出现，或者得到了错误的答案，根据第一条规则，这是一个缺陷。

计算器产品说明书指明计算器不会出现崩溃、死锁或者停止反应等情况，如果测试人员按键后，计算器却停止接收等，根据第二条规则，这是一个缺陷。若在测试过程中发现，由于电池没电等原因导致计算不正确，但产品说明书上没有指出此情况下应该如何进行处理，这也是一个缺陷。

若在进行测试时，发现除了产品说明书规定的加、减、乘、除运算功能之外，还能够进行求平方根的运算，而这一功能并没有在说明书中给出，根据第三条规则，这是一个缺陷。

如果测试人员发现计算器某些功能不好使用，如按键太小、显示屏在亮光下无法看清等，根据第四条规则，这是一个缺陷。

3. 软件缺陷分类

按照 CMM5 中定义的规范，软件缺陷分为多个等级，分别为致命、严重、一般和提示。

(1) 致命。致命性漏洞主要为：内存泄漏、用户数据丢失或被破坏、系统崩溃/死机/冻结、模块无法启动或异常退出、严重的数值计算错误、功能设计与需求严重不符、其它导致无法测试的错误和造成系统不稳定等情况。

(2) 严重。严重性漏洞主要为：功能未实现、功能严重错误、系统刷新错误、语音或数据通信错误、轻微的数值计算错误、系统所提供的功能或服务受明显的影响但不会影响到系统稳定性。

(3) 一般。一般性漏洞主要为：操作界面错误(包括数据窗口内列名定义、含义是否一致)、边界条件下错误、提示信息错误(包括未给出信息、信息提示错误等)、长时间操作无进度提示、系统未优化(性能问题)、光标跳转设置不好、鼠标(光标)定位错误等。

(4) 提示。提示性漏洞主要为：界面格式等不规范、辅助说明描述不清楚、操作时未给出用户提示、可输入区域和只读区域没有明显的区分标志、个别不影响产品理解的错别字、文字排列不整齐等一些小问题及建议。

1.1.4　三种纠错技术

纠错先要查错。查错的工作量通常占整个纠错的 90%以上。下面介绍几种常用的查明程序错误时可能采用的工具和手段，这些方法能明显提高查错的效率。

1) 插入打印语句

在程序中插入暂时性的打印语句，是一种十分常见的查错技术。这类打印语句的作用主要是显示程序的中间结果或有关变量的内容。插入打印语句适用于任何高级语言编写的程序。

2) 设置断点

查错的基本技术之一，就是在程序的可疑区设置断点。每当程序执行到设置的断点时，就会暂停执行，以便纠错者观察变量内容和分析程序的运行状况。

3) 掩蔽部分程序

对可疑程序进行检查时，常常要让程序反复执行。如果整个程序较长，可疑区仅占其中的一小部分，则每次运行整个程序必将浪费许多时间和精力。在这种情况下，往往将不需要检查的程序掩蔽起来，只让可疑的部分程序反复运行。

掩蔽无关程序可使用下述方法：

(1) 在要掩蔽的语句行加上注释符。

(2) 把要掩蔽的程序段置入“常假”的选择结构中，使其无法执行。

1.2　软 件 过 程

1.2.1　RUP

RUP(Rational Unified Process)译为 Rational 统一过程，是 Rational 公司(现归属 IBM 公司)推出的一种软件过程产品。RUP 以统一建模语言(UML)描述软件开发过程，具体如下所示。

1．RUP 各个阶段

RUP 中的软件生命周期在时间上被分解为 4 个顺序的阶段，分别是初始阶段、细化阶段、构建阶段和交付阶段。每个阶段结束于一个主要的里程碑；每个阶段本质上是两个里程碑之间的时间跨度。在每个阶段的结尾执行一次评估以确定这个阶段的目标是否已经满足，如果评估结果满意，则允许项目进入下一个阶段。其中每个阶段又可以进一步分解迭代。一个迭代是一个完整的开发循环，产生一个可执行的产品版本，作为最终产品的一个子集，产品增量式地发展，从一个迭代过程到另一个迭代过程，直到成为最终的系统。

如图 1.1 所示，RUP 可用二维坐标来描述。横轴通过时间组织，是过程展开的生命周期特征，体现开发过程的动态结构，用来描述它的术语主要包括周期、阶段、迭代和里程碑；纵轴以内容来组织，体现开发过程的静态结构，用来描述它的术语主要包括活动、产物、工作者和工作流。

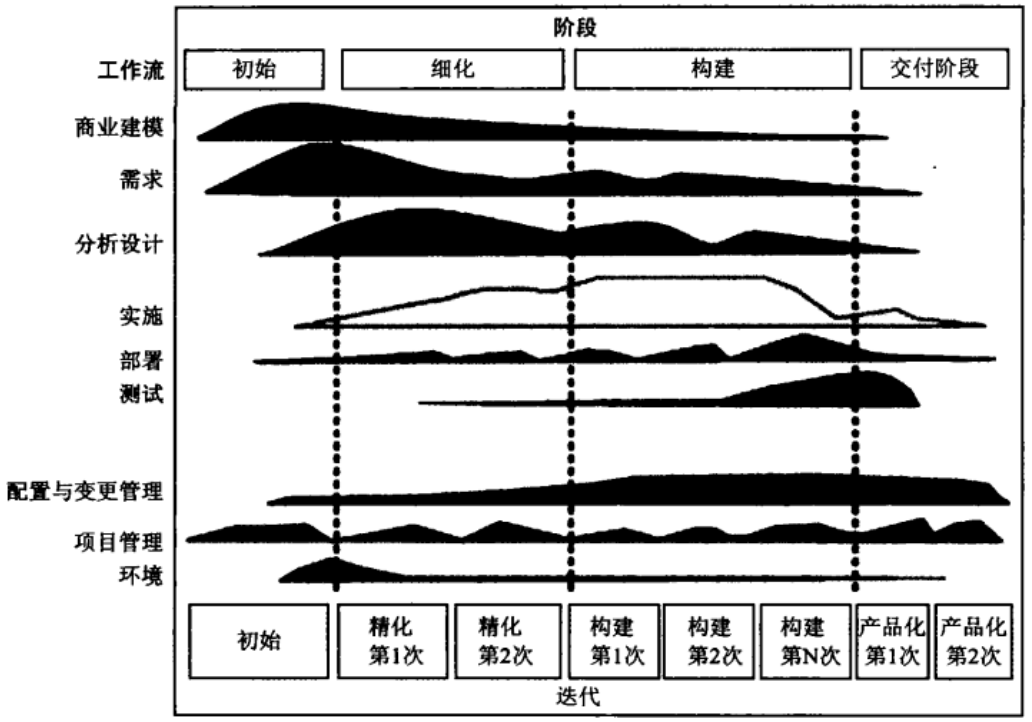


图 1.1 RUP 的过程图

下面依次介绍 RUP 软件生命周期的四个阶段。

1) 初始阶段

初始阶段的目标是为系统建立商业案例并确定项目的边界。为了达到该目标必须识别所有与系统交互的外部实体，并在较高层次上定义交互的特性。这包括识别出所有用例并描述几个重要的用例。本阶段具有非常重要的意义，在这个阶段中所关注的是整个项目进行中的业务和需求方面的主要风险。对于建立在已有系统基础上的开发项目来讲，初始阶段可能很短。

在初始阶段应该取得如下成果：

(1) 蓝图文档，即关于项目的核心需求、关键特性、主约束的总体蓝图。

(2) 初始的用例模型(约占总体的 10%～20%)。

(3) 初始的项目术语表。

(4) 初始的商业案例，包括商业环境、验收标准等。

(5) 初始的风险评估。

(6) 项目计划。

(7) 一个或多个原型。

2) 细化阶段

细化阶段的目标是分析问题领域，建立坚实的体系结构基础，制定项目计划，消除项目中风险最高的因素。为了达到该目标，必须在理解整个系统的基础上，对体系结构作出决策，包括其范围、主要功能和诸如性能等非功能需求，同时为项目建立支持环境，包括创建开发案例，创建模板、工作准则并准备工具。

细化阶段的成果包括：

(1) 用例模型。所有的用例和参与者都已被识别出，并完成大部分的用例描述。

(2) 补充非功能性要求以及与特定用例没有关联的需求。

(3) 软件体系结构的描述。

(4) 可执行的软件原型。

(5) 修订过的风险清单和商业案例。

(6) 整个项目的开发计划，该开发计划应体现迭代过程和每次迭代的评价标准。

(7) 更新的开发案例。

(8) 初步的用户手册。

3) 构建阶段

在构建阶段，组件和应用程序的其余功能被开发并集成为产品，所有的功能都被彻底地测试。从某种意义上说，构建阶段是一个制造过程，其重点为管理资源及控制运作，从而降低成本、加速进度和优化质量。

构建阶段的成果是可以交付给最终用户的产品，它至少包括：

(1) 集成于适当操作系统平台上的软件产品。

(2) 用户手册。

(3) 当前版本的描述。

4) 交付阶段

交付阶段的重点是确保软件对最终用户是可用的。交付阶段可以跨越几次迭代，包括为发布做准备的产品测试，基于用户反馈的少量的调整。在这一阶段，用户反馈应主要集中在产品调整、设置、安装和可用性问题。

2. RUP 核心工作流

RUP 共有 9 个核心工作流，分为 6 个核心过程工作流和 3 个核心支持工作流。6 个核心过程工作流可能使人想起传统瀑布模型中的几个阶段，但应注意迭代过程中的阶段是完全不同的，这些工作流在整个生命周期中被多次执行。

1) 商业建模

商业建模工作流描述了如何为新的目标组织开发一个构想，并基于这个构想在商业用例模型和商业对象模型中定义组织的过程、角色和责任。

2) 需求

需求工作流的目标是描述系统应该做什么，并使开发人员和用户就这一描述达成共识。为了达到该目标，要对需要的功能和约束进行提取、组织、文档化，最重要的是理解系统所解决问题的定义和范围。

3) 分析设计

分析设计工作流是将需求转化成系统的设计，为系统开发一个健壮的结构，使其与实现环境相匹配。设计活动以体系结构设计为中心，其结果是一个设计模型和一个可选的分析模型。设计模型是源代码的抽象，由设计类和一些描述组成。设计类被组织成具有良好接口的设计包和设计子系统，而描述则体现了类的对象如何协同工作实现用例的功能。

4) 实施

实施工作流的目的包括以层次化的子系统形式定义代码的组织结构，以组件的形式(源

文件、二进制文件、可执行文件)实现类和对象，将开发出的组件作为单元进行测试，集成单元使其成为可执行的系统。

5) 测试

测试可通过可靠性、功能性和系统性的三维模型来进行。测试工作流要验证对象间的交互作用，验证软件中所有组件的正确集成，检验所有的需求已被正确的实现，识别并确认缺陷在软件部署之前被提出并处理。RUP 提出的迭代方法是在整个项目中进行测试的，从而尽可能早地发现缺陷，从根本上降低了修改缺陷的成本。

6) 部署

部署工作流的目的是成功地生成版本并将软件分发给最终用户。部署工作流描述了那些与确保软件产品对最终用户具有可用性相关的活动，它包括：软件打包、生成软件本身以外的产品、安装软件、为用户提供帮助。在有些情况下，还可能包括计划和生成 beta 测试版、移植现有的软件和数据以及正式验收。

7) 配置和变更管理

配置和变更管理工作流描绘了如何在多个成员组成的项目中控制大量的软件产物。配置和变更管理工作流提供了准则来管理演化系统中的多个变体，跟踪软件创建过程中的版本。工作流描述了如何管理并行开发、分布式开发，如何自动化创建工程，同时也阐述了对产品修改的原因、时间、人员进行记录，依靠程序员主动、开放、高效的面对面交流来达成对需求、目标、设计实现的理解。

8) 项目管理

软件项目管理平衡各种可能产生冲突的目标，管理风险，克服各种约束并成功交付使用户满意的产品。其目标包括：为项目的管理提供框架，为计划、人员配备、执行和监控项目提供实用的准则，为管理风险提供框架等。

9) 环境

环境工作流的目的是向软件开发组织提供软件开发环境，包括过程和工具。环境工作流集中于配置项目过程中所需要的活动，同样也支持开发项目规范的活动，提供了逐步的指导手册并介绍了如何在组织中实现过程。

3. 用例驱动为核心

开发软件系统的目的是要为该软件系统的用户服务。因此，要创建一个成功的软件系统，必须明白此软件的用户需要什么。“用户”这个术语所指并不仅仅局限于人，还包括其它软件系统。一个用例就是系统向用户提供一个有价值的结果的某项功能。用例是软件的功能性需求。所有用例结合起来就构成了“用例模型”，该模型描述系统的全部功能。用例模型取代了系统的传统的功能规范说明。功能规范说明描述为 “需要该系统做什么？”，而用例驱动则是“需要该系统为每个用户做什么？”因此，用例模型是从用户的利益角度出发进行考虑，设计人员创建一系列用例模型，开发人员审查每个后续模型，以确保它们符合用例模型。测试人员将测试软件系统的实现，以确保实现模型中的组件正确实现了用例。这样，用例不仅启动了开发过程，而且与开发过程结合在一起。

1.2.2 敏捷过程

传统计划驱动的开发方法往往不能获得良好的效果，并且由于过分强调过程控制，因

此在开发过程中产生大量的文档，给开发人员带来很多额外的工作量，因此被称为重量级方法。这种方法使程序员花费大量的开发时间在开发文档的撰写和维护上，而真正花在代码上的时间较少，并且由于依赖过程控制，而不是程序员自我管理，导致开发过程管理复杂且低效，极大地阻碍了软件开发效率。

1. 敏捷开发简介

2001 年是全球软件行业具有历史意义的一年，就在这一年，“敏捷联盟”成立了，并发表了对传统软件开发过程具有颠覆意义的《敏捷宣言》：“通过开发软件和帮助别人开发软件，我们找到了一些更好的开发软件的方式。通过这一工作，我们得出了这些价值：

- 个体和交互　胜过　过程和工具；
- 可以工作的软件　胜过　面面俱到的文档；
- 客户合作　胜过　合同谈判；
- 响应变化　胜过　遵循计划。

也就是说，尽管右边的项也有价值，但我们认为左边的项更有价值。”

从那以后，在敏捷过程价值观的指导下诞生了很多具体的开发过程，包括 XP(极限编程)、FDD(功能驱动开发)、SCRUM 开发过程等等，其中 XP 是影响最为广泛的一种开发过程。

2. 敏捷开发的特征

敏捷方法具有如下两个主要特征：

(1) 开发采用适应性方法，经过多次小型迭代开发过程逐步逼近实际需求，从而为客户提供实际需要的软件。这种开发方法的核心是小型发布，不断集成和严格回归测试。每一次小型发布都经过严格测试后集成到最终产品中，保证每一次小型发布都是经过测试的高质量的代码。在每一次小型发布后和客户沟通，得到客户反馈，不断修改，增加新的客户需要的功能，从而生产出符合客户需要的产品。敏捷开发过程以代码为核心，而不是以文档为核心。传统的以文档为主要输出的设计过程(需求分析、高层架构设计、概要设计、详细设计等)被大大压缩，以最快的速度进入代码的生产过程。设计中以简单为原则，不进行多余的设计活动，小组通过密切而有效率的交流达到对设计的统一理解和认识。文档作为交流工具的作用被弱化，文档作为管理监督的功能被取消。一切以代码为核心，代码经编写、测试、发布、重构，然后进入第二次迭代。

(2) 以人为本。传统计划驱动方法企图以文档、过程为核心，从而抹杀人的重要性。在敏捷方法里，程序员在软件开发中不再是单纯被管理的对象，而是开发的主体。所有的主要设计策略的制定、开发方法的选择、需求的确定都由程序员决定，以往开发过程中的对开发过程的严格控制和检测，以及对软件的各种测量等等都大大简化。

3. 敏捷开发的价值与局限

敏捷方法抛弃了繁琐的文档管理，依靠程序员主动、开放、频繁的面对面的高效交流来达成对需求、目标、设计实现的理解。敏捷方法抛弃了机械、严格的过程控制，依赖程序员和开发团队的高标准自我要求：严格的自律，团队合作精神，个人高度自觉的主动性，责任感。

敏捷方法的高效和高质量实际上是以程序员的高素质和开发团队的高度合作的开发文化为基础的。因此敏捷方法的实施前提是必须找到愿意并有能力实施敏捷方法的团队。XP

的创始人 Beck 也曾建议过有些情况是不适合采用 XP 方法的。比如，不能接受 XP 文化，团队规模过大，重构开销过大等。

1.3 软件质量

1.3.1 概述

软件质量具有多种定义。ANSI/IEEE Std 729—1983 定义软件质量为“与软件产品满足规定的和隐含的需求的能力有关的特征或特性的全体”。CMM 对质量的定义是：① 一个系统、组件或过程符合特定需求的程度；② 一个系统、组件或过程符合客户或用户的要求或期望的程度。M.J.Fisher 定义软件质量为“所有描述计算机软件优秀程度的特性的组合”。

因此，软件质量是一个复杂的多层面概念。

(1) 从用户角度出发，质量是对需求的满足，软件需求是度量软件质量的基础。

(2) 从软件产品角度出发，质量是软件的内在特征。

(3) 从软件开发过程出发，质量是对过程规范的符合。

软件质量框架是由“质量特性—质量子特性—度量因子”构成的 3 层结构模型，如图 1.2 所示。

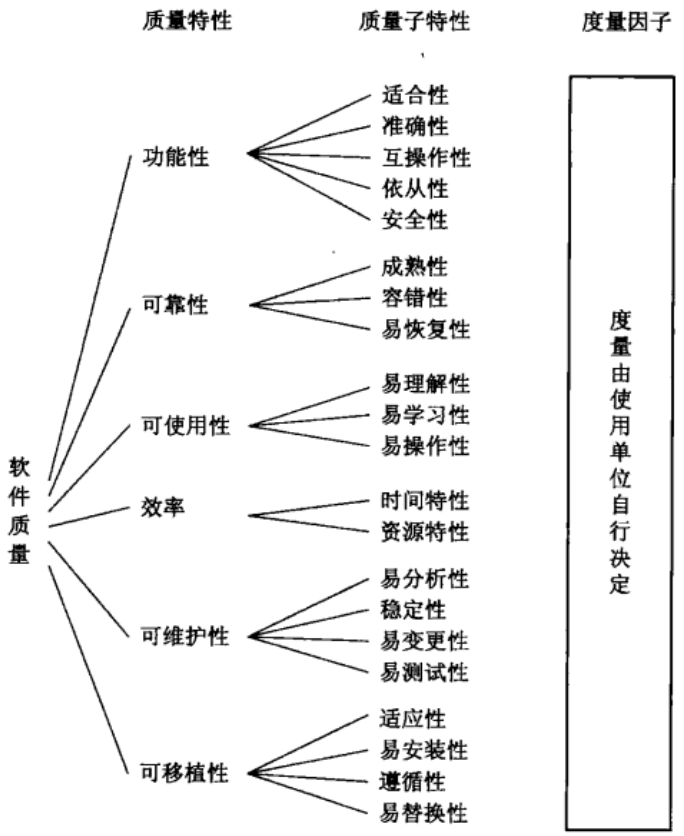


图 1.2 软件质量框架

1.3.2 CMM/CMMI

1987 年 9 月，卡耐基—梅隆大学发布了一份能力成熟度框架以及一套成熟度问卷。4 年后，SEI 推出了 CMM1.0 版。CMM1.0 版在成熟度框架的基础上建立了可用的模型，该模型有效地帮助软件公司建立和实施过程改进计划。近几年，SEI 又推出了 CMM2.0 等版本，同时进入了 ISO 体系，称为 ISO/IEC15504(软件过程评估)。

1. CMM

CMM 是卡耐基—梅隆大学软件工程研究院受美国国防部委托制定的软件过程改良和评估模型，也称为 SEI SW-CMM。该模型于 1991 年发布，其核心是把软件开发视为一个过程，进行过程的监控和研究。CMM 提供了一个软件过程改进的框架，这个框架与软件生命周期无关，与所采用的开发方法无关。

CMM 为软件企业的过程能力提供了一个阶梯式的进化框架，该框架共有 5 级，分别是初始级、可重复级、已定义级、已管理级和优化级。第一级实际上是一个起点，任何准备按 CMM 体系进化的企业都自然处于这个起点上，并通过这个起点向第二级迈进。除第一级外，每一级都设定了一组目标，如果达到了这组目标，则表明达到了这个成熟级别，可以向下一个级别迈进，如表 1.1 所示。

表 1.1 SW-CMM 不同成熟度等级过程的可视性和过程能力

能力等级	特 点	可视性	过程能力
初始级	软件过程是混乱无序的，对过程几乎没有定义，成功依靠的是个人的才能和经验，管理方式属于反应式	有限的可视性	一般达不到进度和成本的目标
可重复级	建立了基本的项目管理来跟踪进度、费用和功能特征，制定了必要的项目管理，能够利用以前类似的项目应用取得成功	里程碑上具有管理可视性	由于基于过去项目的性能，项目开发计划比较现实可行
已定义级	已经将软件管理和过程文档化、标准化，同时综合成该组织的标准软件过程，所有的软件开发都使用该标准软件过程	项目定义软件过程的活动具有可视性	基于已定义的软件过程，组织持续地改善过程能力
已管理级	收集软件过程和产品质量的详细度量，对软件过程和产品质量有定量的理解和控制	定量地控制软件过程	基于对过程和产品的度量，组织持续地改善过程能力
优化级	软件过程的量化反馈和新的思想和技术促进过程的不断改进	不断地改善软件过程	组织持续地改善过程能力

1) *初始级*

在这个阶段，软件开发过程表现得非常随意，偶尔会出现混乱的现象，只有很少的工作过程是经过严格定义的，开发成功往往依靠的是某个人的智慧和努力。此时的软件机构基本没有健全的软件工程管理制度，其软件过程完全取决于项目组的人员配备，具有不可预测性。人员变了过程也随之改变。如果一个项目碰巧由一个杰出的管理者和一支有经验、有能力的开发队伍承担，则这个项目可能是成功的。但是，更常见的情况是，由于缺乏健全的管理和周密的计划，延期交付和费用超支的情况经常发生，结果大多数行动只是应付

危机，而不是完成事先计划好的任务。

总之，处于 1 级成熟度的软件机构，其过程能力是不可预测的，其软件过程是不稳定的，产品质量只能根据相关人员的个人工作能力而不是软件机构的过程能力来预测。

2) 可重复级

这一阶段已经建立了基本的项目管理过程，只需按部就班地设计功能、跟踪费用，根据项目进度表进行开发。对于相似的项目，可以重用以前已经开发成功的部分。处于 2 级成熟度的软件机构，针对所承担的软件项目已建立了基本的软件管理控制制度。通过对以前项目的观察和分析，可以提出针对现行项目的约束条件。项目负责人跟踪软件产品开发的成本和进度以及产品的功能和质量，并且识别出为满足约束条件所应解决的问题。此时，软件的需求是条理化，而且其完整性是受控制的。软件机构已经制定了项目标准，并且能确保严格执行这些标准。项目组与客户及承包商已经建立起一个稳定的、可管理的工作环境。

处于 2 级成熟度的软件机构的过程能力可以概况为：软件项目的策划和跟踪是稳定的，已经为一个有纪律的管理过程提供了可重复以前成功实践的项目环境。软件项目工程活动处于项目管理体系的有效控制之下，执行着基于以前项目的准则且合乎现实的计划。

3) 已定义级

在这一阶段，软件开发的工程活动和管理活动都是文档化、标准化的，是被集成为一个有组织的标准开发过程。所有项目的开发和维护都在这个标准基础上进行定制。处于 3 级成熟度的软件机构，有一个固定的过程小组从事软件过程工程活动。过程小组可以利用过程模型进行过程例化活动，从而获得一个针对某个特定的软件项目的过程实例，并投入过程运作而开展有效的软件项目工程实践。同时，过程小组还可以推进软件机构的过程改进活动。在该软件机构内实施了培训计划，能够保证全体项目负责人和项目开发人员具有完全承担任务所要求的知识和技能。

处于 3 级成熟度的软件机构的过程能力可以概况为：无论是管理活动还是工程活动都是稳定的。软件开发的成本和进度以及产品的功能和质量都受到控制，而且软件产品的质量具有可追溯性。这种能力是基于在软件机构中对已定义的过程模型的活动、人员和职责都有共同的理解。

4) 已管理级

这一阶段已经对软件开发过程和产品质量的测试细节有了很好的归纳，产品和开发过程都可以定量地分解和控制。

处于 4 级成熟度的软件机构的过程能力可以概况为：软件过程是可度量的，软件过程在可度量的范围内运行。这一级的过程能力允许软件机构在定量的范围内预测过程和产品质量趋势，在发生偏离时可以及时采取措施予以纠正，并且可以预期软件产品是高质量的。

5) 优化级

这一阶段通过建立开发过程的定量反馈机制，不断产生新的思想，采用新的技术来优化开发过程。处于 5 级成熟度的软件机构，可以通过对过程实例性能的分析和确定产生某一缺陷的原因，来防止再次出现这种类型的缺陷。通过对任何一个过程实例的分析所获得的经验教训都可以成为该软件机构优化其过程模型的有效依据，从而使其它项目的过程实例得到优化。这样的软件机构可以通过从过程实施中获得的定量的反馈信息，在采用新思想和新技术的同时测试它们，从而不断地改进和优化软件过程。

处于 5 级成熟度的软件机构的过程能力可以概况为：软件过程是可优化的。这一级的软件机构能够持续不断地改进其过程能力，既对现行的过程实例不断地改进和优化，又借助于所采用的新技术和新方法来实现未来的过程改进。

总而言之，根据软件生产的历史与现状，CMM 框架可用 5 个不断进化的层级来表达：其中初始级是混沌的过程；可重复级是经过训练的软件过程；已定义级是标准一致的软件过程；已管理级是可预测的软件过程；优化级是能持续改善的软件过程。任何企业所实施的软件过程，都可能在某一方面比较成熟，在另一方面不够成熟，但总体上必然属于这 5 个层级中的某一个层级。在某个层级内部，也有成熟程度的区别。在一个较低层级的上沿，很可能与一个较高层级的下沿非常接近，此时由这个较低层级向该较高层级进化也就比较容易。反之，在一个较低层级的下沿向较高层级进化，就比较困难。在 CMM 框架的不同层级中，需要解决带有不同层级特征的软件过程问题。因此，一个软件开发企业首先需要了解自己处于哪一个层级，然后才能够对症下药地针对该层级的特殊要求解决相关问题。任何软件开发企业在致力于软件过程改善时，只能由所处的层级向紧邻的上一层级进化，即软件过程的进化是渐进的，而不能是跳跃的。

2. CMMI

CMMI 是 SEI 于 2000 年发布的 CMM 的新版本。CMMI(Capability Maturity Model Integration，能力成熟度模型集成)将各种能力成熟度模型，包括 Software CMM、Systems Eng-CMM、People CMM 和 Acquisition CMM 等，整合到同一架构中去，由此建立包括软件工程、系统工程和软件采购等在内的模型集成，以解决除软件开发以外的软件系统工程和软件采购工作中的需求。

软件企业采用多种模型改进过程能力时，往往会存在以下一些问题：

(1) 软件企业不能集中不同过程改进的能力以取得更大成绩。

(2) 软件企业往往要进行一些重复的培训、评估和改进活动，从而增加了成本。

(3) 由于不同模型有些说法不一致，或活动不协调，甚至相抵触，导致难以理解。

正因为如此，美国国防部把各种能力成熟度模型集成为 CMMI，其基本思想如下：

(1) 解决软件项目过程改进难度增大问题。

(2) 实现软件工程的并行与多学科组合。

(3) 实现过程改进的最佳效益。

CMMI 具有如下两个功能：第一，软件采购方法的改革；第二，从集成产品与过程的角度出发，包含健全的系统开发原则的过程改进。

CMMI 纠正了 CMM 存在的一些缺点，消除了不同模型之间的不一致和重复，降低了基于模型改善的成本，指导组织改善软件过程，提高了产品和服务的开发、获取和维护能力。CMMI 更加适用企业的过程改进实施。需要注意的是，SEI 并没有废除 CMM 模型，只是停止了 CMM 评估方法。

1.3.3　质量与测试

软件测试和软件质量是分不开的。测试是手段，质量是目的。软件测试作为一种辅助而且必需的手段，客观反映某个时间段内的软件质量。测试人员执行软件测试，发现软件

BUG，反馈给开发人员进行修改，然后经过再次测试，以确认该 BUG 已被解决。测试人员重复此过程直至软件符合最终用户需求。在 BUG 产生之前，BUG 的具体数量、位置、出现时间等都是未知数，测试人员根本就不知道。

测试可以发现 BUG，但不能避免 BUG 的产生。QA 团队对项目进行近似完全的控制，通过建立标准和方法论，有条理地仔细监视和评价软件开发过程，对发现的软件缺陷提出解决建议，执行某些测试，从而从源头上防止 BUG 的出现。

实现软件质量保证主要有两种途径：一种是通过贯彻软件工程各种有效的技术方法和措施使得尽量在软件开发期间减少错误；另一种是通过分析和测试软件来发现和纠正错误。

软件测试属于软件控制，作为软件质量的重要保证，其和质量保证有如下 4 点区别，如表 1.2 所示。

表 1.2 软件测试与软件质量保证的区别

项 目	软件测试	质量保证
工作性质	技术性工作	管理性工作
对象	软件产品(包括阶段性产品)	软件过程
焦点	事后检验	强调预防
范围	研发部门和技术部门	公司层面，包括所有部门

通过实施 CMM/CMMI，有助于改进软件产品的质量，改进项目满足预定目标能力，减少开发成本和周期，降低项目风险，提高组织过程能力，提高市场占有率。实施不同等级的 CMM/CMMI，对于按照每功能点来计算的软件缺陷率的降低具有显著的作用，如表 1.3 所示。

表 1.3 实施 CMM 对软件质量的影响

CMM 等级	隐含的缺陷(每功能点)	缺陷消除率	交付的缺陷
1	5.00	85%	0.75
2	4.00	89%	0.44
3	3.00	91%	0.27
4	2.00	93%	0.14
5	1.00	95%	0.05

CMM/CMMI 基于过程的软件质量管理主要包括质量保证和质量控制两大方面。软件质量保证是由(相对)独立的质量管理人员在项目的整个开发周期中对项目所执行的过程和产生的工作产品进行监督和检查，确保其符合预定的要求。软件质量保证的目的是确保过程得到有效的执行及推进过程改进，并就项目过程的执行情况和所构造的产品向管理者提供适当的可视性。软件质量控制是指为评价和验证已开发的产品而执行的活动和技术，包括验证产品是否满足质量要素的要求，以及产品(包括生命周期的工作产品)是否具有接受的质量。

软件质量控制所采取的主要技术是软件测试。通过软件测试，来验证产品是否符合技术文档预期的特性、功能和性能等要求，并识别产品的缺陷。

虽然目前在 CMM/CMMI 中没有明确软件测试作为一个独立的过程域，但是在 CMM/CMMI 很多地方都涉及了软件测试内容。CMMI 和软件测试最主要的区别在于：CMMI 是对于软件过程的改进(包括软件测试过程)；而软件测试是针对项目结果的验证，在测试时是从侧面改进软件质量的。例如，在 CMMI 的“确认”和“验证”两个过程域中，CMMI

列出了以下实践。

验证特定目标及实践摘要

SG 1　确认准备
- SP 1.1　选择需确认之产品
- SP 1.2　选择确认环境
- SP 1.3　选择确认程序与准则

SG 2　确认产品或产品组件
- SP 2.1　执行确认
- SP 2.2　分析确认结果

测试计划
　　测试范围识别、测试准备、工作流程、测试准入与完成准则

SG 1　验证准备
- SP 1.1　选择需验证之工作产品
- SP 1.2　选择验证环境
- SP 1.3　选择验证程序与准则

SG 2　执行同行审查
- SP 2.1　准备同行审查
- SP 2.2　进行同行审查
- SP 2.3　分析同行审查资料

SG 3　验证工作产品
- SP 3.1　执行验证
- SP 3.2　分析验证结果

测试执行
　　缺陷记录、缺陷分析、缺陷修改与回归测试报告

1.4　测试与开发的关系

软件开发与软件测试具有密不可分的关系，我们从以下几个方面进行介绍。

1．测试与开发各阶段的关系

图 1.3 给出了软件测试与软件开发各阶段的关系，软件测试在开发阶段具有如下作用。

(1) 项目规划阶段：负责从单元测试到系统测试的整个测试阶段的监控。

(2) 需求分析阶段：确定测试需求分析、系统测试计划的制定，评审后成为管理项目。

(3) 详细设计和概要设计阶段：确保集成测试计划和单元测试计划完成。

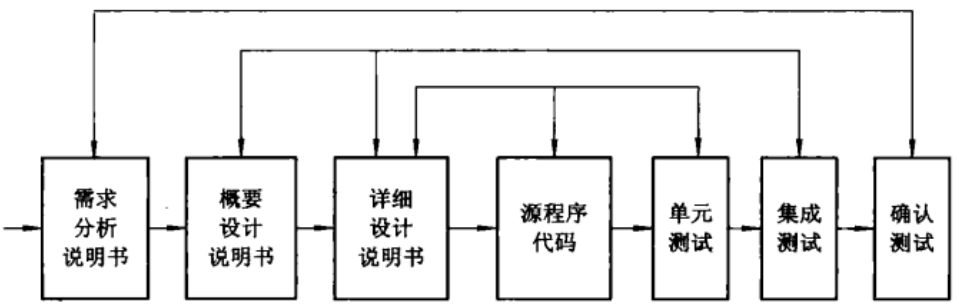


图 1.3　软件测试与软件开发过程的关系

(4) 编码阶段：由开发人员完成自己负责部分的测试代码。当编写工作项目较大时，由专人进行编码阶段的测试任务。

(5) 测试阶段(单元、集成、系统测试)：依据测试代码进行测试，并提交相应的测试状态报告和测试结束报告。

2. 测试与开发的并行性

图 1.4 给出了软件测试与软件开发之间的并行关系。

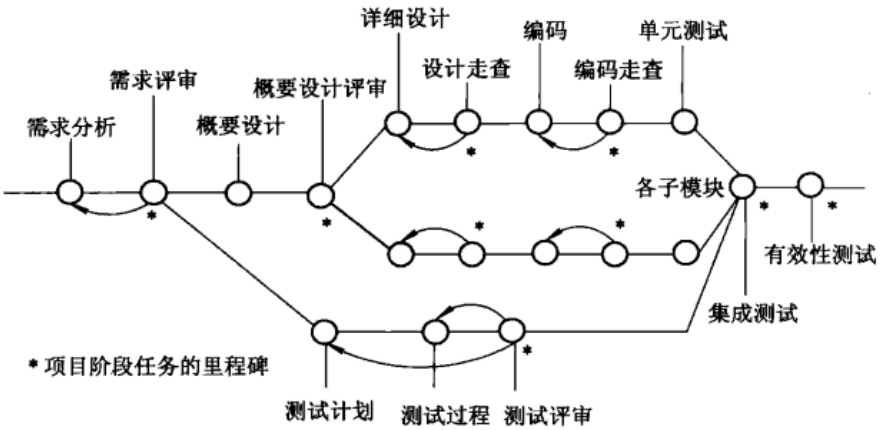


图 1.4 软件测试与软件开发的并行性

3. 完整的软件开发流程

从图 1.5 中可以看出，测试过程和开发过程贯穿软件过程的整个生命周期，二者相辅相成，相互依赖，具体概括为如下 3 个关键点：

(1) 测试过程和开发过程是同时开始、同时结束的，两者保持同步的关系。

(2) 测试过程是对开发过程中阶段性成果和最终产品进行验证的过程，所以两者相互依赖。前期，测试过程更多地依赖于开发过程；而后期，开发过程更多地依赖于测试过程。

(3) 测试工作的重点和开发工作的重点可能是不一样的，两者有各自的特点。

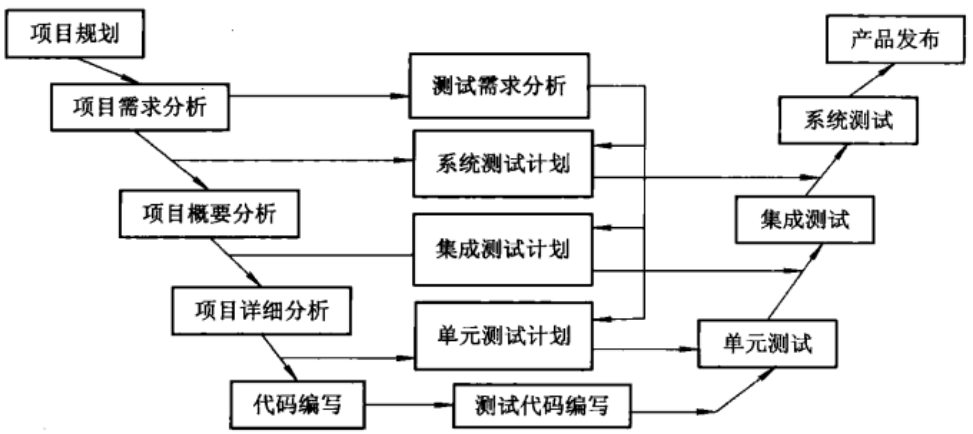


图 1.5 完整的软件开发流程

1.5 思考与习题

一、选择题

1．软件本身的特点和目前软件开发模式使隐藏在软件内部的质量缺陷不可能完全避免，在下列关于导致软件质量缺陷的原因的描述中，不正确的是_____。

A. 软件需求模糊以及需求的变更，从根本上影响着软件产品的质量

B. 目前广为采用的手工开发方式难于避免出现差错

C. 程序员编码水平低下是导致软件缺陷的最主要原因

D. 软件测试技术具有缺陷

2．缺陷产生的原因是(　　)。

A. 交流不充分及沟通不畅；软件需求的变更；软件开发工具的缺陷

B. 软件的复杂性；软件项目的时间压力

C. 程序开发人员的错误；软件项目文档的缺乏

D. 以上都是

二、判断题

1. 缺乏有力的方法学指导和有效的开发工具的支持，往往是产生软件危机的原因之一。(　　)

2. 目前的绝大多数软件都不适合于快速原型技术。(　　)

3. 在程序运行之前没法评估其质量。(　　)

三、简答题

1. 什么是软件？软件发展经过了哪几个阶段？

2. RUP 是什么？具有什么特征？

3. 软件质量与软件测试是什么关系？

4. 软件质量框架是什么？包括什么内容？

5. CMM 是什么？其具体内容是什么？CMMI 与 CMM 的关系是什么？

6. 软件测试与软件开发具有什么关系？

第2章 软件测试基本知识

本章主要介绍软件测试发展历程、软件测试目的和原则以及分类，并就软件测试模型和测试用例等理论知识给出了详细讲解。

2.1 软件测试发展历程

软件测试伴随着软件的产生而产生。早期软件开发过程中，软件规模小，复杂程度低，软件开发过程相当混乱无序，软件测试含义也比较窄，等同于“调试”。此时软件测试的目的是纠正软件的故障，常常由软件开发人员自己进行。对测试的投入极少，测试介入也晚，常常是等到形成代码、产品已经基本完成时才进行测试。

1957年，软件测试首次作为发现软件缺陷的活动，与调试区分开来。1972年，北卡罗来纳大学举行首届软件测试会议，John Good Enough 和 Susan Gerhart 在 IEEE 上发表的《测试数据选择的原理》确定了软件测试是软件的一种研究方向。1979年，Glenford Myers 在《软件测试艺术》一书中提出“测试是为发现错误而执行的一个程序或者系统的过程”。

20世纪80年代早期，软件和IT行业进入了大发展，软件向大型化、高复杂度的方向发展，软件的质量越来越重要。一些软件测试的基础理论和实用技术开始形成，软件开发的方式也逐渐由混乱无序的开发过程过渡到结构化的开发过程。以结构化分析与设计、结构化评审、结构化程序设计以及结构化测试为特征，软件测试性质和内容也随之发生变化，不再是一个单纯的发现错误的过程，而是具有软件质量评价的内容。1983 年，Bill Hetzel 在《软件测试完全指南》中指出，测试是以评价一个程序或者系统属性为目标的一种活动，是对软件质量的度量。IEEE 给软件测试定义为“使用人工或自动手段来运行或测定某个软件系统的过程，其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别”。这个定义明确地指出，软件测试的目的是为了检验软件系统是否满足需求，软件测试不再是一个一次性的活动，也不只是开发后期的活动，而是与整个开发流程融为一体的。

20世纪90年代，软件测试工具开始运用。1996年，测试支持度 TSM、测试成熟度 TMM 等一系软件测试相关理论被提出。到了2002年，Rick 和 Stefan 在《系统的软件测试》一书中对软件测试做了进一步描述：测试是为了度量和提高软件的质量，对软件进行工程设计、实施和维护的整个生命周期过程。

近20年来，随着计算机和软件技术的飞速发展，软件测试技术的研究也取得了很大的突破。许多测试模型(如 V 模型等)产生，单元测试、自动化测试等方面涌现了大量的软件测试工具。在软件测试工具方面，商业化的软件测试工具，如捕获/回放工具、Web 测试工具、性能测试工具、测试管理工具、代码测试工具等大量涌现，一些开放源码社区中也出现了许多软件测试工具，这些工具得到了广泛应用且相当成熟和完善。

2.2　软件测试目的

软件测试是指使用人工或者自动手段来运行或测试某个系统的过程，其目的在于检验被测试系统是否满足规定的要求或弄清预期结果与实际结果之间的差别。

软件测试是帮助识别开发完成(中间或最终的版本)计算机软件(整体或部分)的正确度、完全度和质量的软件过程，是软件质量保证的重要子域。

Grenford J.Myers 曾对软件测试的目的提出过以下观点:

(1) 测试是为了证明程序有错，而不是证明程序无错误;

(2) 一个好的测试用例在于它能发现至今未发现的错误;

(3) 一个成功的测试是指发现了至今未发现的错误的测试。

这种观点指出测试是以查找错误为中心，而不是为了演示软件的正确功能。但是只从字面意思理解可能会产生误导，认为发现错误是软件测试的唯一目的，查找不出错误的测试就是没有价值的测试。

软件测试的目的往往包含如下内容:

(1) 测试并不仅仅是为了找出错误，而且要通过分析错误产生的原因和错误的发生趋势，帮助项目管理者发现当前软件开发过程中的缺陷，以便及时改进。

(2) 测试分析帮助测试人员设计出有针对性的测试方法，以改善测试的效率和有效性。

(3) 没有发现错误的测试也是有价值的，完整的测试是评定软件质量的一种方法。

测试的目标就是以最少的时间和人力找出软件中潜在的各种错误和缺陷，证明软件的功能和性能与需求说明相符。此外，实施测试收集到的测试结果数据也为可靠性分析提供了依据。

2.3　软件测试原则

软件测试应遵循以下基本原则:

(1) 应当把“尽早地和不断地进行软件测试”作为软件开发者的座右铭。由于软件的复杂性和抽象性，软件开发各个阶段工作的多样性，以及参加开发各种层次人员之间工作的配合关系等因素，使得开发的每个环节都可能产生错误。因此不应把软件测试仅仅看做是软件开发的一个独立阶段，而应当把它贯穿到软件开发的各个阶段中。坚持在软件开发的各个阶段进行技术评审，尽早发现和预防错误，把出现的错误克服在早期，杜绝某些隐患，提高软件质量。

(2) 测试用例应由测试输入数据和与之对应的预期输出结果两部分组成。测试用例主要用来检验程序员编制的程序，不但需要测试输入数据，而且需要针对这些输入数据的预期输出结果。如果对测试输入数据没有得到预期的程序输出结果，那么就缺少了检验实测结果的基准，就有可能把一个似是而非的错误结果当成正确结果。

(3) 程序员应避免检查自己的程序。测试工作需要严谨的作风、客观的态度和冷静的情绪。心理学告诉我们，人们具有一种不愿否定自己工作的心理，而这一心理状态就成为测

试自己程序的障碍。另外，程序员对软件规格说明理解错误而引入的错误则更难发现。因此由别人来测试程序员编写的程序，可能会更客观、更有效，并更容易取得成功。

(4) 在设计测试用例时，应当包括合理的输入条件和不合理的输入条件。合理的输入条件是指能验证程序正确的输入条件；而不合理的输入条件是指异常的、临界的、可能引起问题异变的输入条件。用不合理的输入条件测试程序时，往往比用合理的输入条件进行测试能发现更多的错误。

(5) 充分注意测试中的群集现象。经验表明，测试后程序中残存的错误数目与该程序中已发现的错误数目或检错率成正比。根据这个规律，应当对错误群集的程序段进行重点测试，以提高测试投资的效益。

在所测程序段中，若发现错误数目多，则残存错误数目也比较多。这种错误群集现象已为许多程序的测试实践所证实。例如 IBM 公司的 OS/370 操作系统中，大量的错误仅与该系统的 4%的程序模块有关。因此，如果发现某一程序模块似乎比其它程序模块有更多的错误倾向时，则应当花费较多的时间和代价测试这个程序模块。

(6) 严格执行测试计划，排除测试的随意性。测试计划应包括：所测软件的功能，输入和输出，测试内容，各项测试的进度安排，资源要求，测试资料，测试工具，测试用例的选择，测试的控制方式和过程，系统组装方式，跟踪规程，调试规程，回归测试的规定以及评价标准等。

(7) 应当对每一个测试结果做全面检查。有些错误的征兆在输出实测结果时已经明显地出现了，但是如果不仔细、全面地检查测试结果，就会使这些错误被遗漏掉。所以必须对预期的输出结果明确定义，对实测的结果仔细分析检查，抓住征候，暴露错误。

(8) 妥善保存测试计划、测试用例、出错统计和最终分析报告，为软件维护提供方便。

2.4 软件测试分类

2.4.1 按照开发阶段划分

软件测试贯穿整个软件开发的始末，按照软件开发阶段划分，软件测试分为单元测试、集成测试、确认测试、系统测试、验收测试等。

2.4.2 按照执行主体划分

按照测试实施组织划分，软件测试分为开发方测试、用户测试和第三方测试。

1. 开发方测试

开发方测试通常也叫“验收测试”或“α 测试”。在软件开发环境中，开发者检测与证实软件的实现是否满足软件设计说明或软件需求说明的要求。用户测试是指在用户的应用环境下，用户检测与核实软件实现是否符合自己预期的要求。

2. 用户测试

通常用户测试被称为 β 测试，指把软件有计划地、免费地分发到目标市场，让用户大量使用、评价和检查软件。通常情况下，用户测试不是指用户的“验收测试”，而是指用户

的使用性测试，由用户在应用过程中发现软件的缺陷与问题，并对使用质量进行评价。

3．第三方测试

第三方测试是指由第三方测试机构来进行的测试，也称独立测试。第三方测试由在技术、管理和财务上与开发方和用户方都相对独立的组织进行，一般在模拟用户真实应用的环境下，进行软件的确认测试。

2.4.3　按照执行状态划分

按照测试执行状态划分，软件测试分为动态测试和静态测试。

1．静态测试

静态测试是指计算机不真正运行被测试的程序，而是人工对程序和文档进行分析与检查，包括走查、符号执行、需求确认等。静态测试一方面利用计算机作为对被测程序进行特性分析的工具，与人工测试有着根本的区别；另一方面并不真正运行被测程序，与动态方法也不相同。因此，静态测试常称为“分析”，是对被测程序进行特性分析方法的总称。

静态分析过程如图 2.1 所示。其中，针对代码的静态测试包括代码检查、静态结构分析、代码质量度量等。

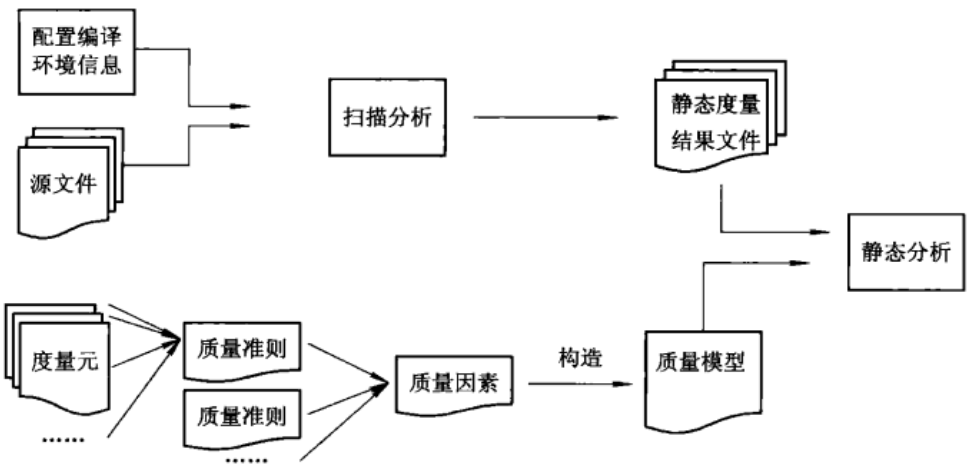


图 2.1　静态分析过程

1) 代码检查

代码检查主要检查代码和设计的一致性，代码对文档标准的遵循及代码的可读性，代码的逻辑表达正确性，代码结构的合理性等方面。代码检查比动态测试更有效率，能快速找到 30%～70%的逻辑设计错误和代码缺陷。

代码检查一般在编译和动态测试之前进行，其实施方法很多，如走查(Walk through)、审查(Inspection)、评审(Review)等，如表 2.1 所示。

(1) 走查。走查是在开发组内部进行，通过个人检查和阅读等手段来查找错误。其主要检查逻辑错误和代码是否符合标准、规范和风格等，具有不在现场修改等特点。

(2) 审查。审查是在开发组内部进行，通过分配相关的角色，采用讲解、提问并使用检查表方式进行的查找错误活动。审查通过会议的形式进行，一般参加人员包括与本模块相

关的开发人员，由一名开发者讲解、其它开发者提问，本模块开发者回答问题、填写检查表。

(3) 评审。评审是由开发组、测试组和相关人员(QA、产品经理等)联合进行，采用讲解、提问并使用检查表方式进行的查找错误的活动。评审一般有正式的计划、流程和结果报告。其中，同行(对等)评审是指由与工作产品开发人员具有同等背景和能力的人员对工作产品进行的评审，其目的是有效地消除软件的缺陷。

表 2.1 走查与会议评审的对比

事 项	走 查	会 议 评 审
准备	通读设计和编码	应准备好需求描述文档、程序设计文档、程序的源代码清单、代码编码标准和代码缺陷检查表
形式	非正式会议	正式会议
参加人员	开发人员为主	项目组成员，包括测试人员
主要技术方法	无	缺陷检查表
注意事项	限时，不要现场修改代码	限时，不要现场修改代码
生成文档	会议记录	静态分析错误报告
目标	代码标准规范，无逻辑错误	代码标准规范，无逻辑错误

采用静态分析技术进行代码检查具有如下优点：

(1) 测试人员直接面对的是问题的本身而不是征兆。

(2) 代码检查可以发现其它方法无法发现的逻辑错误。

(3) 代码检查的效率是最初测试效率的 3～5 倍。

(4) 代码检查可以发现 75%～80%的错误。

2) *静态结构分析*

静态结构分析以图形的方式表现程序的内部结构，例如，函数调用关系图、函数内部控制流图等。其中，函数调用关系图描述程序中函数调用与被调用的关系，控制流图显示函数的逻辑结构。

2．动态测试

动态测试指通过运行被测程序，检查运行结果与预期结果的差异，并分析运行效率和健壮性等性能的测试方法。这种方法由三部分组成：构造测试实例、执行程序和分析程序的输出结果。

2.4.4 按照测试技术划分

按照对被测对象的了解程度划分，软件测试分为黑盒测试、白盒测试和灰盒测试。

1．黑盒测试

黑盒测试也称功能测试或数据驱动测试。它是在已知产品所应具有的功能的前提下，通过测试来检测每个功能是否都能正常使用。在测试时，把程序看做一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，测试者在程序接口进行测试，只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而

产生正确的输出信息，并且保持外部信息(如数据库或文件)的完整性。

黑盒测试试图发现以下类型的错误：功能错误或遗漏、界面错误、数据结构或外部数据库访问错误、性能错误、初始化和终止错误等。

2．白盒测试

白盒测试又称结构测试或逻辑驱动测试。与黑盒测试正好相反，它是知道产品内部工作过程，检测产品内部动作是否按照规格说明书的规定正常进行，按照程序内部的结构测试程序，检验程序中的每条通路是否都能按预定要求正确工作。白盒测试的主要方法有逻辑驱动、路径测试等，主要用于软件验证。

白盒测试是基于源代码的测试，需要了解程序的构架、具体需求以及一些编写程序的技巧，能够检查一些程序规范、指针、变量、数组越界等问题。

白盒测试容易发现以下类型的错误：变量没有声明、无效引用、数组越界、死循环、函数本身没有析构、参数类型不匹配、调用系统的函数没有考虑到系统的兼容性等。

3．灰盒测试

灰盒测试介于黑盒测试和白盒测试之间，主要用于测试各个组件之间的逻辑关系是否正确，相对白盒测试来说要求相对较低，对测试用例要求也相对较低，用于代码的逻辑测试、验证程序接收和处理参数。灰盒测试的重点在于测试程序的处理能力和健壮性，相对黑盒测试和白盒测试而言，投入的时间相对少，维护量也较小。

软件测试方法与软件开发过程相关联。单元测试一般应用白盒测试方法，集成测试应用灰盒测试方法，系统测试和确认测试应用黑盒测试方法。

黑盒测试和白盒测试比较如表 2.2 所示。

表 2.2　黑盒测试和白盒测试比较

项　目	黑盒测试法	白盒测试法
规划方面	功能测试	结构测试
性质	是一种确认技术，回答“我们在构造一个正确的系统吗？”(Verification)	是一种验证技术，回答“我们在正确地构造一个系统吗？”(Validation)
优点	(1) 确保从用户角度出发 (2) 适用于各阶段测试 (3) 从产品功能角度测试 (4) 容易入手生成测试数据	(1) 针对程序内部特定部分进行覆盖测试 (2) 可构成测试数据使特定程序部分得到测试 (3) 有一定的充分性度量手段 (4) 可获得较多工具支持
缺点	(1) 无法测试程序内部特定部分 (2) 某些代码得不到测试 (3) 如果规格说明有误，则无法发现 (4) 不易进行充分性测试	(1) 无法测试程序外部特性 (2) 不易生成测试数据(通常) (3) 无法对未实现规格说明的部分进行测试 (4) 工作量大，通常只用于单元测试
应用范围	边界分析法、等价类划分法、决策表测试	语句覆盖、判定覆盖、条件覆盖、路径覆盖等

2.4.5 按照软件发布范围划分

为了满足全球化软件在世界范围内发布的需要，在全球化软件的开发过程中，软件的国际化设计和本地化工程处理是两个重要的步骤。与开发过程相对应，作为软件质量保证过程的全球性软件测试过程，包含一系列相互关联的测试技术和流程，可以分为国际化测试、本地化能力测试和本地化测试等阶段。

1. 国际化测试

国际化测试的目的是测试软件的国际化支持能力，发现软件的国际化的潜在问题，保证软件在世界不同区域都能正常运行。以 Windows 应用软件为例，针对世界软件市场的语言优先级，需要首先应用德语和日语的操作系统。这两种语言代表最重要的区域市场，同时日语又作为东亚双字节字符的典型语言，在英文 Windows 操作系统上安装具体的语言支持文件进行区域设置。将日语作为系统默认区域设置进行测试，可验证 ANSI(非 Unicode)组件中的双字节字符集(DBCS)处理。将德语作为系统默认区域设置进行测试，可确保需要进行文本转换时能够正确处理 ANSI 和 OEM 代码页。

国际化测试使用每种可能的国际输入类型，针对任何区域性或区域设置检查产品的功能是否正常。软件国际化测试的重点在于执行国际字符串的输入/输出功能。国际化测试数据必须包含东亚语言、德语、复杂脚本字符和英语(可选)的混合字符，其中复杂脚本字符指阿拉伯语、希伯来语、泰语。国际化测试中发现的比较严重的软件错误包括软件在不同区域设置环境下的功能丢失或数据破坏。这些错误经常出现在字符编码转换和双字节字符的输入/输出过程中。

2. 本地化能力测试

本地化能力是指不需要重新设计或修改代码，将程序的用户界面翻译成任何目标语言的能力。本地化能力高的软件可以容易地实施本地化处理。本地化能力测试的目的是测试软件的本地化支持能力，尽早发现软件本地化时将会出现的潜在错误。本地化能力测试通过以后，表示产品已可用于本地化，才能进行软件的本地化过程和本地化测试。为了降低本地化能力测试的成本，提高测试效率，本地化能力测试通常在软件的伪本地化版本上进行。软件的伪本地化是指将软件中需要本地化的英文文本，使用其它本地化的文本替换，模拟本地化版本的过程。

本地化能力测试中发现的典型错误包括：字符的硬编码(即软件中需要本地化的字符写在了代码内部)，对需要本地化的字符长度设置了固定值，在软件运行时以控件位置定位，图标和位图中包含了需要本地化的文本，软件的用户界面与文档术语不一致等。

3. 本地化测试

本地化测试的目的是测试特定目标区域设置的软件本地化质量。本地化测试的对象是软件的本地化版本。本地化测试的环境是在本地化的操作系统上安装本地化的软件。从测试方法上分为基本功能测试、安装/卸载测试、当地区域的软硬件兼容性测试。测试的内容主要包括软件本地化后的界面布局和软件翻译的语言质量，包含软件、文档和联机帮助等部分。

本地化测试的错误主要包括软件用户界面错误，如布局错误(版式、大小和位置)，本地化有关的功能错误，翻译错误和双字节支持错误。软件的翻译质量包括翻译的准确性、完

整性、一致性以及特定区域市场的文化、传统、习俗和政治的敏感内容。为了保证软件本地化测试的质量和成本，通常外包给当地语言为母语的本地化服务公司。

总之，全球化软件的测试是保证软件全球发布的质量保证活动。软件的国际化测试、本地化能力测试和本地化测试是一系列互相联系的测试过程。遵循国际化设计和保持良好的本地化能力，可以保证软件的国际化和本地化测试，实现软件的全球化和本地化。

2.5　软件测试模型

2.5.1　V 模型

V 模型作为最典型的测试模型，由 Paul Rook 在 20 世纪 80 年代后期提出，如图 2.2 所示。V 模型反映了测试活动与分析和设计的关系，明确标明了测试过程中存在的不同级别，并清楚描述测试的各个阶段和开发过程的各个阶段之间的对应关系。V 模型左侧是开发阶段，右侧是测试阶段。开发阶段先从定义软件需求开始，然后把需求转换为概要设计和详细设计，最后形成程序代码。测试阶段是在代码编写完成以后，先从单元测试开始，然后是集成测试、系统测试和验收测试。

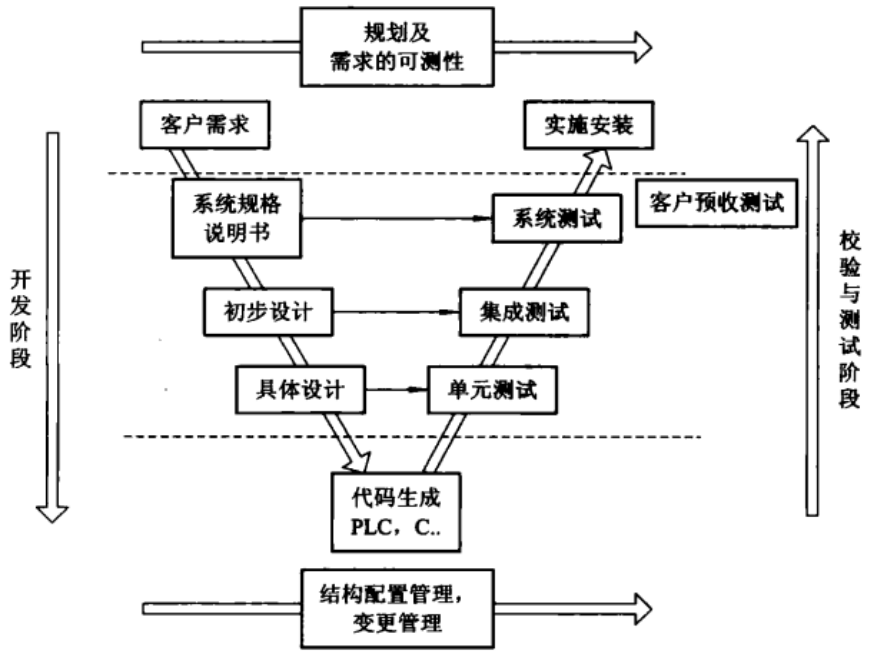


图 2.2　V 模型示意图

单元测试对应详细设计，也就是说，单元测试用例和详细设计文档一起实现；而集成测试对应于概要设计，其测试用例是根据概要设计中模块功能及接口等实现方法编写。依次类推，测试计划在软件需求完成后就开始进行，完成系统测试用例的设计等。

V 模型存在如下一些局限性：它仅把测试过程作为在需求分析、概要设计、详细设计及编码之后的一个阶段，主要针对程序进行寻找错误的活动，而忽视了测试活动对需求分析、系统设计等活动的验证和确认的功能。

2.5.2 W 模型

相对于 V 模型而言，W 模型增加了软件各开发阶段中应同步进行的验证和确认(V&V)活动。如图 2.3 所示，W 模型由两个 V 字型模型组成，分别代表测试与开发过程，明确表示出了测试与开发的并行关系。

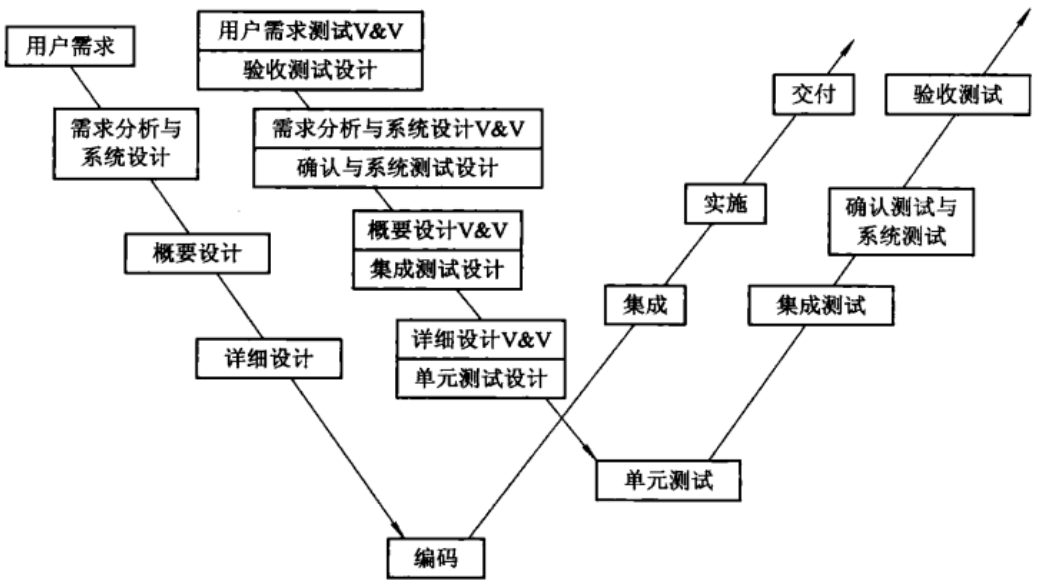


图 2.3 W 模型示意图

W 模型强调，测试伴随着整个软件开发周期，测试的对象不仅仅是程序，需求、设计等同样要测试，也就是说，测试与开发同步进行。W 模型有利于尽早地发现问题，只要相应的开发活动完成，就可以开始测试。例如，需求分析完成后，测试就应该参与到对需求的验证和确认活动中，以尽早地找出缺陷所在。同时，对需求的测试也有利于及时了解项目难度和测试风险，及早制定应对措施，从而减少总体测试时间，加快项目进度。

W 模型也存在局限性。在 W 模型中，需求、设计、编码等活动被视为串行，测试和开发活动保持着一种线性的前后关系，上一阶段结束，才开始下一个阶段工作，因此，W 模型无法支持迭代开发模型。

2.5.3 H 模型

V 模型和 W 模型都认为软件开发是需求、设计、编码等一系列串行的活动，而事实上，这些活动在大部分时间内可以交叉。因此，相应的测试也不存在严格的次序关系，单元测试、集成测试、系统测试之间具有反复迭代。正因为 V 模型和 W 模型存在这样的问题，H 模型将测试活动完全独立出来，使得测试准备活动和测试执行活动清晰地体现出来。

图 2.4 仅仅显示了整个测试生命周期中某个层次的“微循环”。H 模型揭示了软件测试作为一个独立的流程贯穿于软件整个生命周期，与其它流程并发地进行，并指出软件测试要尽早准备，尽早执行。不同的测试活动可以按照某个次序先后进行，也可能是反复的，只要某个测试达到准备就绪点，测试执行活动就可以开展。因此，H 模型具有如下意义：

(1) 测试准备与测试执行分离，有利于资源调配，降低成本，提高效率。

(2) 充分体现测试过程(不是技术)的复杂性。

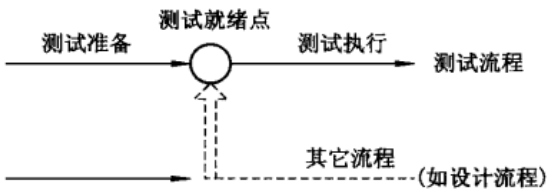


图 2.4　H 模型示意图

2.5.4　X 模型

由于 V 模型没能体现出测试设计、测试回溯的过程，因此出现了 X 测试模型。

如图 2.5 所示，X 模型的左边描述的是针对单独程序片段所进行的编码和测试，此后将进行频繁的交接，通过集成最终合成为可执行的程序。X 模型右上方定位了已通过集成测试的成品进行封版并提交给用户，也可以作为更大规模和范围内集成的一部分。多根并行的曲线表示变更可以在各个部分发生。X 模型右下方定位了探索性测试。这是不进行事先计划的特殊类型的测试，往往帮助有经验的测试人员在测试计划之外发现软件错误。

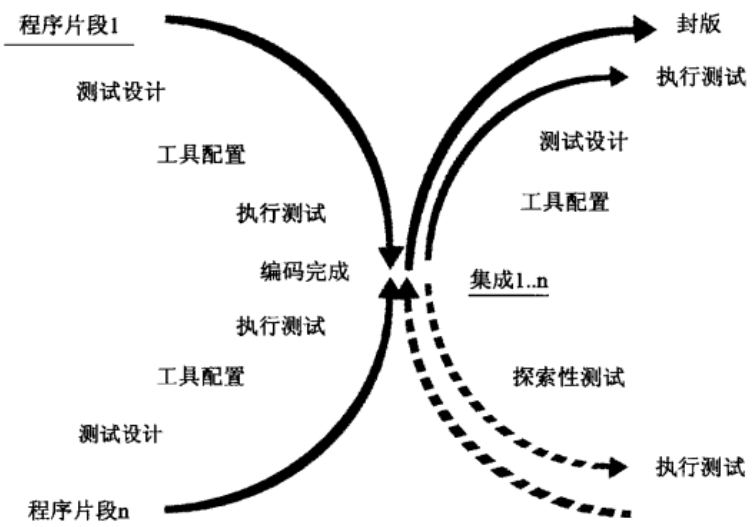


图 2.5　X 模型示意图

2.5.5　前置模型

前置模型将测试和开发紧密结合，具有如下的优点。

(1) 开发和测试相结合。前置测试模型将开发和测试的生命周期整合在一起，标识了项目生命周期从开始到结束之间的关键行为，表示这些行为在项目周期中的价值。

前置测试在开发阶段以“编码→测试→编码→测试”的方式进行。也就是说，程序片段编写完成后会进行测试。

(2) 对每一个交付内容进行测试。每一个交付的开发结果都必须通过一定的方式进行测试。源程序代码并不是唯一需要测试的内容。可行性报告、业务需求说明以及系统设计文档等也是被测试的对象。这同 V 模型中开发和测试的对应关系相一致，并且在其基础上有所扩展。

(3) 让验收测试和技术测试保持相互独立。验收测试应该独立于技术测试，这样可以提供双重的保险，以保证设计及程序编码能够符合最终用户的需求。验收测试既可以在实施阶段的第一步来执行，也可以在开发阶段的最后一步执行。

(4) 反复交替的开发和测试。项目开发中存在很多变更，例如需要重新访问前一阶段的内容，或者跟踪并纠正以前提交的内容，修复错误，增加新发现的功能等。开发和测试需要一起反复交替地执行。

(5) 引入新的测试理念。前置测试对软件测试进行优先级划分，用较低的成本及早发现错误，并且充分强调了测试对确保系统高质量的重要意义。

2.6 测 试 用 例

2.6.1 测试用例的基本概念

测试用例作为测试工作的指导，是软件测试必须遵守的准则，是软件测试质量稳定的根本保障。测试用例(Test Case)目前没有经典的定义。简单地说，测试用例就是设计一个情况，软件程序在这种情况下，必须能够正常运行并且达到程序所设计的执行结果。比较通常的说法是：指对一项特定的软件产品进行测试任务的描述，体现测试方案、方法、技术和策略，内容包括测试目标、测试环境、输入数据、测试步骤、预期结果、测试脚本等，并形成文档。

测试用例是将软件测试的行为活动做了一个科学化的组织归纳，目的是将软件测试的行为转化成可管理的模式，同时测试用例也是将测试具体量化的方法之一。

测试用例的重要性体现在以下几方面。

(1) 测试用例构成了设计和制定测试过程的基础。

(2) 测试的“深度”与测试用例的数量成比例，这是由于每个测试用例反映不同的场景、条件或经由产品的事件流。

(3) 测试工作量与测试用例的数量成比例。根据全面且细化的测试用例，可以更准确地估计测试周期各连续阶段的时间安排。

(4) 测试设计和开发的类型以及所需的资源主要受控于测试用例。

(5) 测试用例通常根据它们所关联的测试类型或测试需求来分类，而且将随类型和需求进行相应的改变。最佳方案是为每个测试需求至少编制两个测试用例：一个测试用例用于证明该需求已经满足，通常称做正面测试用例；另一个测试用例反映某个无法接受、反常或意外的条件或数据，用于论证只有在所需条件下才能够满足该需求，这个测试用例称做负面测试用例。

测试用例主要有如下几种分类。

(1) 功能测试用例：包含功能测试、健壮性测试和可靠性测试。
(2) 性能测试用例：包含性能测试、压力测试和强度测试。
(3) 集成测试用例：包含接口测试、健壮性测试和可靠性测试。
(4) 安全测试用例。
(5) 用户界面测试用例及少量功能测试用例。
(6) 安装/反安装测试用例。

测试种类、阶段和用例的关系如表 2.3 所示。

表 2.3　测试种类阶段和用例关系列表

测试阶段	测 试 类 型	执 行 人 员
单元测试	模块功能测试，包含部分接口测试、路径测试	开发人员
集成测试	接口测试、路径测试，含部分功能测试	开发人员，如果测试人员水平较高可以由测试人员执行
系统测试	功能测试、健壮性测试、性能测试、用户界面测试、安全性测试、压力测试、可靠性测试、安装/反安装测试	测试人员
验收测试	对于实际项目基本同上，并包含文档测试；对于软件产品主要测试相关技术文档	测试人员，可能包含用户

2.6.2　测试用例的编写

编写测试用例文档应有文档模板，须符合内部的规范要求。测试用例文档将受制于测试用例管理软件的约束。

测试用例文档由简介和测试用例两部分组成。简介部分编制了测试目的、测试范围、定义术语、参考文档、概述等。测试用例部分逐一列示各测试用例。每个具体测试用例都将包括下列详细信息：用例编号、用例名称、测试等级、入口准则、验证步骤、期望结果(含判断标准)、出口准则、注释等。以上内容涵盖了测试用例的基本元素：测试索引、测试环境、测试输入、测试操作、预期结果、评价标准。表 2.4 所示是一个典型的测试用例文档。

表 2.4　测试用例文档

编制人		审定人		时间	
软件名称			编号/版本		
测试用例					
用例编号					
参考信息(参考的文档及章节号或功能项)：					
输入说明(列出选用的输入项，列出预期输出)：					
输出说明(逐条与输入项对应，列出预期输出)：					
环境要求(测试要求的软硬件、网络要求)：					
特殊规程要求：					
用例间的依赖关系：					
用例产生的测试程序限制：					

测试用例可根据基本事件、备选事件和异常事件设计相关的用例。设计基本事件的用例，应该参照用例规约(或设计规格说明书)，根据关联的功能、操作按路径分析法设计测试用例。而对孤立的功能则直接按功能设计测试用例。基本事件的测试用例应包含所有需要实现的需求功能，覆盖率达100%。

设计备选事件和异常事件的用例，则要复杂许多。

2.6.3 测试用例的作用

1. 指导测试的实施

测试用例主要适用于集成测试、系统测试和回归测试。在实施测试时测试用例作为测试的标准，测试人员一定要按照测试用例严格按用例项目和测试步骤逐一实施测试，并将测试情况记录在测试用例管理软件中，以便自动生成测试结果文档。

根据测试用例的测试等级，集成测试应测试哪些用例，系统测试和回归测试又该测试哪些用例，在设计测试用例时都已作明确规定，实施测试时测试人员不能随意作变动。

2. 规划测试数据的准备

按照测试用例配套准备一组或若干组测试原始数据，以及标准测试结果。尤其像测试报表之类数据集的正确性，按照测试用例规划准备测试数据是十分必要的。

除正常数据之外，还必须根据测试用例设计大量边缘数据和错误数据。

3. 编写测试脚本的“设计规格说明书”

为提高测试效率，软件测试已大力发展自动测试。自动测试的中心任务是编写测试脚本。如果说软件工程中软件编程必须有设计规格说明书，那么测试脚本的设计规格说明书就是测试用例。

4. 评估测试结果的度量基准

完成测试实施后需要对测试结果进行评估，并且编制测试报告。判断软件测试是否完成、衡量测试质量需要一些量化的结果。例如：测试覆盖率是多少，测试合格率是多少，重要测试合格率是多少，等等。

5. 分析缺陷的标准

通过收集缺陷，对比测试用例和缺陷数据库，分析确认是漏测还是缺陷复现。漏测反映了测试用例的不完善，应立即补充相应测试用例，最终达到逐步完善软件质量的目的。而已有相应测试用例，则反映实施测试或变更处理存在问题。

2.6.4 相关问题

1. 测试用例的评审

测试用例是软件测试的准则，但它并不是一经编制完成就成为准则。测试用例在设计编制过程中要组织同级互查。完成编制后应组织专家评审，需获得通过才可以使用。评审委员会可由项目负责人、测试、编程、分析设计等有关人员组成，也可邀请客户代表参加。

2. 测试用例的修改更新

测试用例在形成文档后还需要不断完善。主要来自三方面的缘故：第一，在测试过程中发现设计测试用例时考虑不周，需要完善；第二，在软件交付使用后反馈的软件缺陷，而缺陷又是因测试用例存在漏洞造成的；第三，软件自身新增功能以及软件版本的更新，测试用例也必须配套修改更新。

一般小的修改完善可在原测试用例文档上修改，但文档要有更改记录。如果软件的版本升级更新，则测试用例一般也应随之编制升级更新版本。

3. 测试用例的管理软件

运用测试用例还需配备测试用例管理软件。它的主要功能有三个：第一，能将测试用例文档的关键内容，如编号、名称等自动导入管理数据库，形成与测试用例文档完全对应的记录；第二，可供测试实施时及时输入测试情况；第三，最终实现自动生成测试结果文档，包含各测试度量值、测试覆盖表及测试通过或不通过的测试用例清单列表。

2.7 思考与习题

一、选择题

1. 软件测试按照测试技术划分为(　　)。

A. 性能测试、负载测试、压力测试　　B. 恢复测试、安全测试、兼容测试

C. A 与 B 都是　　D. 单元测试、集成测试、验收测试

2. 软件测试的目的是(　　)。

A. 避免软件开发中出现错误

B. 发现软件开发中出现的错误

C. 尽可能发现并排除软件中潜藏的错误，提高软件的可靠性

D. 修改软件中出现的错误

3. 各个地方对软件测试定义不同，请你根据软件测试方面、理论方面、代码的角度测试填空。

代码方面分为：_____、集成测试、系统测试、验收测试。

理论方面分为：_____、动态测试、静态测试

测试方面分为：_____、压力测试、回归测试、负载测试、恢复测试、安全性测试、兼容性测试、内存泄露测试、比较测试等。

A. 单元测试　　B. 黑盒测试　　C. 白盒测试　　D. 负载测试

二、判断题

(1) Beta 测试是验收测试的一种。(　　)

(2) 尽量用公共过程或子程序去代替重复的代码段。(　　)

(3) 测试是为了验证该软件是否已正确地实现了用户的要求。(　　)

(4) 对于连锁型分支结构，若有 n 个判定语句，则有 2n 条路径。(　　)

(5) 尽量采用复合的条件测试，以避免嵌套的分支结构。(　　)

(6) GOTO 语句概念简单，使用方便，在某些情况下，保留 GOTO 语句反能使写出的

程序更加简洁。(　　)

(7) 发现错误多的程序模块，残留在模块中的错误也多。(　　)

三、简答题

1. 软件测试的目的是什么？
2. 软件测试中应注意哪些事项？
3. 按执行主体划分，软件测试分哪几类？
4. 按执行主体划分，软件测试分哪几类？
5. V 模型和 W 模型各自的优缺点是什么？
6. 测试用例是什么？

第 3 章　软件测试过程

本章详细介绍软件测试的整个过程，包括单元测试、集成测试、确认测试和验收测试等。

3.1　软件测试流程概述

软件测试流程与软件开发流程类似，也包括测试计划、测试设计、测试开发、测试执行和测试评估几个部分。

(1) 测试计划。根据用户需求报告中关于功能要求和性能指标的规格说明书，定义相应的测试需求报告，使得随后所有的测试工作都将围绕着测试需求来进行。同时，适当选择测试内容，合理安排测试人员、测试时间及测试资源等。

(2) 测试设计。测试设计是指将测试计划阶段制订的测试需求分解、细化为若干个可执行的测试过程，并为每个测试过程选择适当的测试用例，保证测试结果的有效性。

(3) 测试开发。建立可重复使用的自动测试过程。

(4) 测试执行。执行测试开发阶段建立的自动测试过程，并对所发现的缺陷进行跟踪管理。测试执行一般由单元测试、集成测试、确认测试以及回归测试等步骤组成。

(5) 测试评估。结合量化的测试覆盖域及缺陷跟踪报告，对应用软件的质量和开发团队的工作进度及工作效率进行综合评价。

在上述测试流程中，测试执行按以下步骤进行：单元测试、集成测试、确认测试和验收测试，如图 3.1 所示。

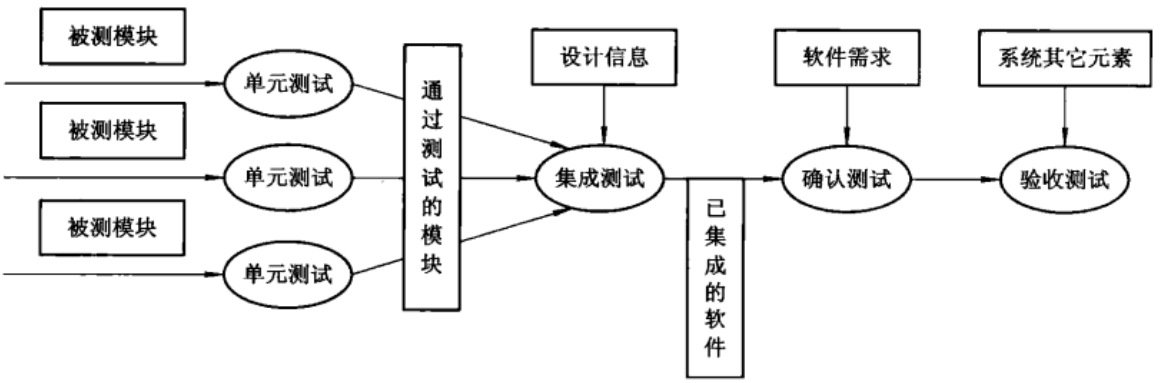


图 3.1　软件测试执行过程

(1) 单元测试：通过对每个最小的软件模块进行测试，对源代码的每一个程序单元实行测试，检查各个程序模块是否正确地实现了规定的功能，确保其能正常工作。

(2) 集成测试：对已测试过的模块进行组装集成，目的在于检验与软件设计相关的程序结构问题。

(3) 确认测试：检验软件是否满足需求规格说明中的功能和性能需求，确定软件配置完全、正确。同时检验软件产品能否与实际运行环境中整个系统的其它部分(如硬件、数据库及操作人员)协调工作。

(4) 验收测试：作为检验软件产品质量的最后一道工序，主要让用户对软件进行测试，并重新执行已经做过的测试的某个子集，保证没有引入新的错误。

3.2 单元测试

单元测试用于判断一小段代码在某个特定条件(或者场景)下某个特定函数的行为，主要测试软件设计的最小单元(模块)在语法、格式和逻辑等方面的缺陷以及是否符合功能需求。程序的多个模块可以并行地进行测试工作。

1. 单元测试内容

单元测试针对程序模块进行测试，主要有以下 5 个任务：模块接口测试、局部数据结构测试、边界条件测试、执行路径测试和出错处理测试，如图 3.2 所示。

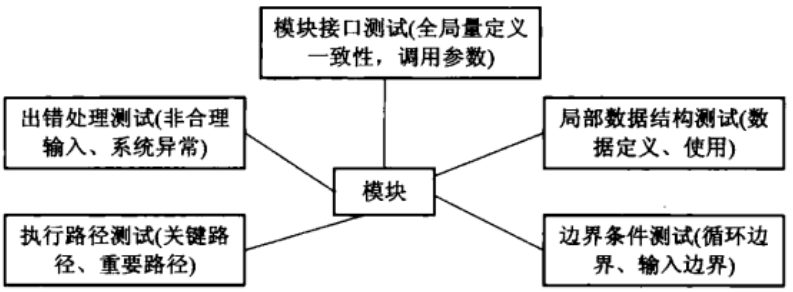


图 3.2 单元测试要解决的任务

1) 模块接口测试

通过对被测模块的数据流进行测试，检查进出模块的数据是否正确。因此，必须对模块接口，包括参数表、调用子模块的参数、全程数据、文件输入/输出等操作进行测试。具体涉及以下内容。

(1) 模块接受输入的实际参数个数与模块的形式参数个数是否一致。

(2) 输入的实际参数与模块的形式参数的类型是否匹配。

(3) 输入的实际参数与模块的形式参数所使用的单位是否一致。

(4) 调用其它模块时，所传送的实际参数个数与被调用模块的形式参数的个数是否相同。

(5) 调用其它模块时，所传送的实际参数与被调用模块的形式参数的类型是否匹配。

(6) 调用其它模块时，所传送的实际参数与被调用模块的形式参数的单位是否一致。

(7) 调用内部函数时，参数的个数、属性和次序是否正确。

(8) 在模块有多个入口的情况下，是否引用与当前入口无关的参数。

(9) 是否修改了只读型参数。

(10) 全局变量是否在所有引用它们的模块中都有相同的定义。

如果模块内包括外部I/O，还应该考虑下列因素:

(1) 文件属性是否正确。

(2) OPEN与CLOSE语句是否正确。

(3) 缓冲区容量与记录长度是否匹配。

(4) 在进行读/写操作之前是否打开了文件。

(5) 在结束文件处理时是否关闭了文件。

(6) 正文书写/输入有无错误。

(7) I/O错误是否检查并做了处理。

2) 局部数据结构测试

测试用例检查局部数据结构的完整性，如数据类型说明、初始化、缺省值等方面的问题，并测试全局数据对模块的影响。

在模块工作过程中，必须测试模块内部的数据能否保持完整性，包括内部数据的内容、形式及相互关系不发生错误。

局部数据结构应注意以下几类错误：不正确的或不一致的类型说明；错误的初始化或默认值；错误的变量名，如拼写错误或书写错误；下溢、上溢或者地址错误。

3) 执行路径测试

测试用例对模块中重要的执行路径进行测试，其中对基本执行路径和循环进行测试往往可以发现大量路径错误。测试用例必须能够发现由于计算错误、不正确的判定或不正常的控制流而产生的错误。

常见的错误如下：误解的或不正确的算术优先级；混合模式的运算错误；错误的初始化；精确度不够精确；表达式的不正确符号表示。

针对判定和条件覆盖，测试用例能够发现如下错误：不同数据类型的比较错误；不正确的逻辑操作或优先级；应当相等的地方由于精确度的错误而不能相等；不正确的判定或不正确的变量；不正确的或不存在的循环终止；当遇到分支循环时不能退出；不适当地修改循环变量。

4) 出错处理测试

检查模块的错误处理功能是否包含错误或缺陷。例如，是否拒绝不合理的输入，出错的描述是否难以理解，是否对错误定位有误，是否出错原因报告有误，是否对错误条件的处理不正确，在对错误处理之前错误条件是否已经引起系统的干预等。

测试出错处理的重点是当模块在工作中发生错误时，其中的出错处理设施是否有效。

检验程序中的出错处理可能面临的情况有:

(1) 对运行发生的错误描述难以理解。

(2) 所报告的错误与实际遇到的错误不一致。

(3) 出错后，在错误处理之前就引起系统的干预。

(4) 例外条件的处理不正确。

(5) 提供的错误信息不足，以至于无法找到错误的原因。

5) 边界条件测试

边界条件测试是单元测试的最后一步，必须采用边界值分析方法来设计测试用例。测

试在为限制数据处理而设置的边界处，测试模块是否能够正常工作。

一些与边界有关的数据类型，如第一个、最后一个、最大值、最小值、最长、最短、最高、最低等。

在边界条件测试中，应设计测试用例检查以下情况：

(1) 在 n 次循环的第 0 次、1 次……n 次是否有错误。

(2) 运算或判断中取最大值、最小值时是否有错误。

(3) 数据流、控制流中刚好等于、大于、小于确定的比较值时是否出现错误。

2. 单元测试步骤

单元测试通常在编码阶段进行。在源程序代码编制完成并经过评审和验证，确认没有语法错误之后，开始设计单元测试的测试用例。

模块并不是一个独立的程序，在考虑测试模块时，同时要考虑它和外界的联系，因此使用一些辅助模块去模拟与被测模块相关的其它模块。辅助模块分为以下两种：

(1) 驱动模块(Drive)：用来模拟被测模块的上一级模块，相当于被测模块的主程序，用于接收测试数据，并把这些数据传送给被测模块，启动被测模块，最后输出实测结果。

(2) 桩模块(Stub)：用来模拟被测模块工作过程中所调用的模块。桩模块一般只进行很少的数据处理，不需要把子模块的所有功能都带进来，但不允许什么事情也不做。

被测模块、与它相关的驱动模块及桩模块共同构成了一个“测试环境”，如图 3.3 所示。

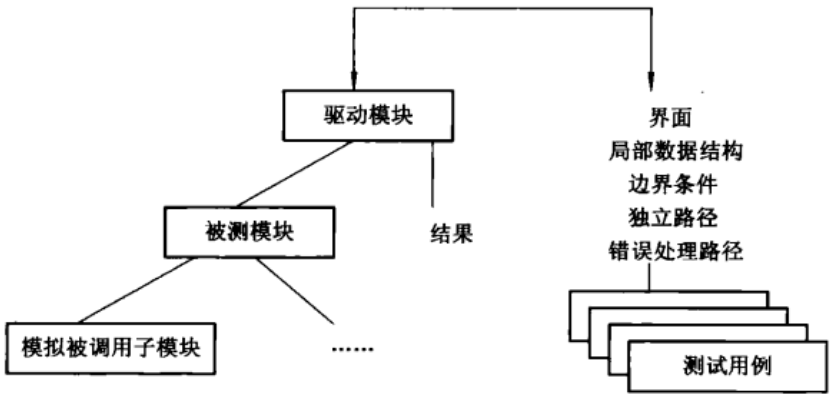


图 3.3 单元测试的测试环境

3.3 集 成 测 试

按照软件设计要求，将经过单元测试的模块连接起来，组成所规定的软件系统的过程称为“集成”。集成测试就是针对这个过程，按模块之间的依赖接口关系图进行的测试。图 3.4 给出了软件分层结构的示例图。

由于集成测试不是在真实环境下进行，而是在开发环境或是一个独立的测试环境下进行的，因此集成测试所需人员一般从开发组中选出，在开发组长的监督下进行。开发组长负责保证在合理的质量控制和监督下使用合适的测试技术执行充分的集成测试。在集成测试过程中，由一个独立测试观察员来监控测试工作。

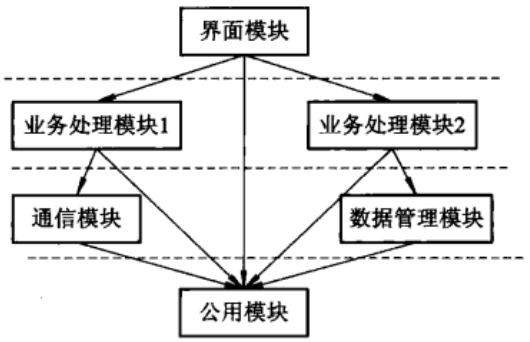


图 3.4　软件分层结构的示意图

1. 集成测试的主要任务

集成测试是组装软件的系统测试技术之一，按设计要求把通过单元测试的各个模块组装在一起之后，要求软件系统符合实际软件结构，发现与接口有关的各种错误。集成测试主要适应于如下几种软件系统：

(1) 对软件质量要求较高的软件系统，如航天软件、电信软件、系统底层软件等。

(2) 使用范围比较广、用户群数量较大的软件。

(3) 使用类似 C/C++带有指针的程序语言开发的软件。

(4) 类库、中间件等产品。

集成测试的主要任务是解决以下 5 个问题：

(1) 将各模块连接起来，检查模块相互调用时数据经过接口是否丢失。

(2) 将各个子功能组合起来，检查能否达到预期要求的各项功能。

(3) 一个模块的功能是否会对另一个模块的功能产生不利的影响。

(4) 全局数据结构是否有问题，会不会被异常修改。

(5) 单个模块的误差积累起来是否被放大，从而达到不可接受的程度。

2. 集成测试方法

集成测试主要测试软件的结构问题，因此测试建立在模块接口上，多为黑盒测试，适当辅以白盒测试。执行集成测试应遵循如下步骤。

步骤 1：确认组成一个完整系统的模块之间的关系。

步骤 2：评审模块之间的交互和通信需求，确认模块间的接口。

步骤 3：生成一套测试用例。

步骤 4：采用增量式测试，依次将模块加入到系统并测试这个过程按逻辑/功能顺序重复进行。

集成测试过程中尤其要注意关键模块测试，关键模块一般具有下述一个或多个特征：

(1) 同时对应几项需求功能；

(2) 具有高层控制功能；

(3) 复杂，易出错；

(4) 有特殊的性能要求。

集成测试的主要目的是验证组成软件系统的各模块的接口和交互作用，因此集成测试对数据的要求从难度和内容上不是很高。集成测试一般不使用真实数据，可以使用一部分

代表性的测试数据。在创建测试数据时，应保证数据充分测试软件系统的边界条件。

集成测试包括非增量式集成测试和增量式集成测试。

1) 非增量式集成测试方法

非增量式集成测试方法采用一步到位的方法来进行测试，对所有模块进行个别的单元测试后，按程序结构图将各模块连接起来，把连接后的程序当作一个整体进行测试。

2) 增量式集成测试方法

增量式集成测试方法具体包括自顶向下增量式测试、自底向上增量式测试以及三明治集成测试。

(1) 自顶向下增量式测试。自顶向下增量式测试按结构图自上而下进行逐步集成和逐步测试。模块集成的顺序是首先集成主控模块(主程序)，然后按照软件控制层次结构向下进行集成。自顶向下的集成方式可以采用深度优先策略和广度优先策略，如图 3.5 所示。

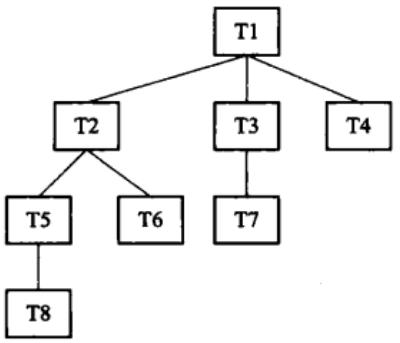


图 3.5 自顶向下增量式测试示意图

图 3.5 中，深度优先顺序为：T1->T2->T5->T8->T6->T3->T7->T4；而广度优先顺序为：T1->T2->T3->T4->T5->T6->T7->T8。

该方法由下列步骤实现。

步骤 1：以主模块为所测试模块兼驱动模块，而所有直属于主模块的下属模块全部用桩模块替换，并对主模块进行测试。

步骤 2：采用深度优先或广度优先测试方法，用实际模块替换相应桩模块，再用桩模块代替它们的直接下属模块，从而与已经测试的模块或子系统组装成新的子系统。

步骤 3：进行回归测试排除组装过程中的错误可能性。

步骤 4：判断是否所有的模块都已经组装到了系统中。如果是，则结束测试，否则转到步骤 2 执行。

自顶向下增量式测试方法的优点如下：

① 在测试过程中较早地验证主要的控制点。

② 功能性的模块测试可以较早地得到证实。

③ 最多只需要一个驱动模块就可进行测试。

④ 支持缺陷故障隔离。

自顶向下增量式测试方法具有如下缺点：

① 随着底层模块的不断增加，会导致底层模块的测试不充分，特别是被重用的模块。

② 由于每次组装都必须提供桩，会使得桩的数目急剧增加，从而使维护桩的成本也快速上升。

因此，该方法适用于大部分采用结构化编程方法，而且软件的结构相对比较简单的情况。

(2) 自底向上增量式测试。自底向上增量式测试是从“原子”模块(软件结构中最底层的模块)开始，按结构图自下而上逐步进行集成和测试。图 3.6 表示采用自底向上增量式测试的过程。

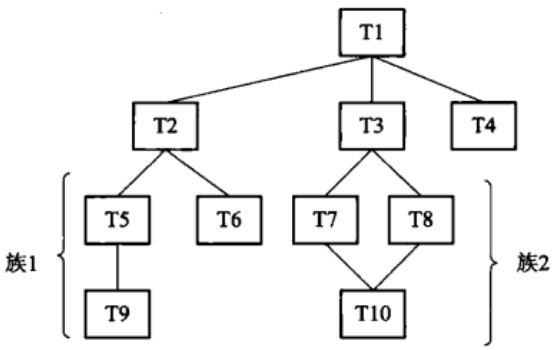


图 3.6　自底向上增量式测试示意图

该方法由下列几个步骤实现。

步骤 1：把底层模块组合成实现某个特定的软件子功能的族。

步骤 2：写一个驱动程序(用于测试的控制程序)，协调测试数据的输入和输出。

步骤 3：对由模块组成的子功能族进行测试。

步骤 4：去掉驱动程序，沿软件结构由下向上移动，把子功能族组合成更大的功能族。

步骤 5：不断重复步骤 2～步骤 4，直到完成。

自底向上增量式测试方法的优点如下：

① 虽然模拟中断或异常需要设计一定的桩模块，但总体上减少了桩模块的工作量。

② 允许对底层模块行为进行早期验证。

③ 在测试初期，可以并行进行集成，相应地比使用自顶向下的方式效率高。

自底向上增量式测试方法具有如下缺点：

① 随着集成到顶层，整个系统变得越来越复杂，对于底层的一些模块将很难覆盖。

② 驱动模块的开发工作量很大。

(3) 三明治集成测试。三明治集成也称混合集成，它将自顶向下和自底向上的优点和缺点集于一身。三明治集成是把系统分为三层，中间一层为目标层。测试时对目标层上面的一层采用自顶向下的集成测试方式，而对目标层下面的一层使用自底向上的集成策略，最后再对目标层进行测试。

如图 3.7 所示，使用程序桩 S2、S3 和 S4 对用户界面 M1 进行测试，使用驱动程序 D5 和 D6 对最底层功能模块 M7、M8 和 M9 进行测试。当整个系统集成时，将程序桩 S2、S3 和 S4 换成中间层模块 M2、M3 和 M4，驱动程序 D5 和 D6 对换成中间层模块 M5 和 M6，从而对中间层的功能模块进行测试。

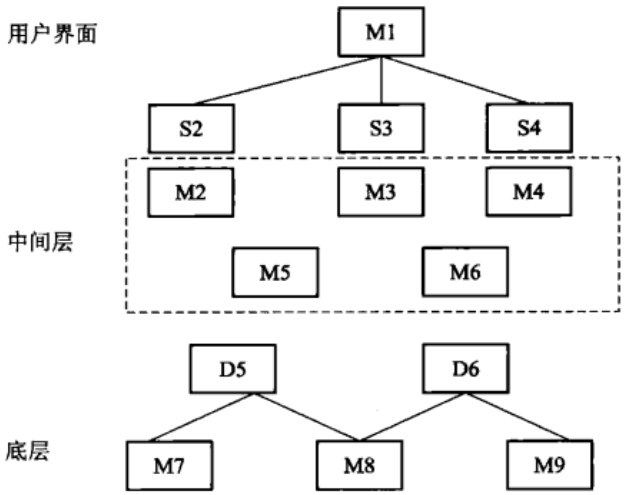


图 3.7 三明治集成测试示意图

表 3.1 给出了各集成测试方法的对比结果。

表 3.1 集成测试方法的比较

名 称	自顶向下增量式	自底向上增量式	三明治集成
集成	早	早	早
基本程序工作时间	早	晚	早
需要驱动程序	否	是	是
需要桩程序	是	否	是
工作并行性	低	中	中
特殊路径测试	难	容易	中等
计划与控制	难	容易	难

下面给出非增量式集成和增量式集成的比较结果。

(1) 非增量式集成测试模式是先分散测试，然后集中起来一次完成集成测试。如果在模块的接口处存在错误，则会在最后的集成测试时一下子暴露出来。非增量式集成测试时可能发现很多错误，但为每个错误定位和纠正非常困难，并且在改正一个错误的同时又可能引入新的错误，从而更难断定出错的原因和位置。与此相反，增量式集成测试采用逐步集成和逐步测试的方法，测试的范围逐步增大，从而使错误易于定位和纠正。因此，增量式集成测试比非增量式集成测试有比较明显的优越性。

(2) 自顶向下测试的主要优点在于自然地做到逐步求精，从一开始就让测试者了解系统的框架。它的主要缺点是需要提供被调用模拟子模块，被调用模拟子模块可能不能反映真实情况，因此测试有可能不充分。

(3) 自底向上测试的优点在于，由于驱动模块模拟了所有调用参数，因此测试数据没有困难。其主要缺点在于，只有到最后一个模块被加入之后才能知道整个系统的框架。

(4) 三明治集成测试采用自顶向下、自底向上集成相结合的方式，并采取持续集成策略，有助于尽早发现缺陷，提高工作效率。

(5) 核心系统先行集成测试能保证一些重要功能和服务的实现，对于快速软件开发有

效。采用此种模式的测试，要求系统能明确区分核心软件部件和外围软件部件，从而可以采用高频集成测试，借助于自动化工具实现。

总之，采用自顶向下集成测试和自底向上集成测试方案较为常见。在实际测试工作中，应该结合项目的实际环境及各测试方案适用的范围进行合理的选型。

3.4 确 认 测 试

确认测试用于验证软件的有效性，即验证软件的功能和性能及其它特性是否与用户的要求一致。

在确认测试阶段需要做的工作如图 3.8 所示。首先要进行有效性测试以及软件配置复审，然后进行验收测试和安装测试，在通过了专家鉴定之后，才能成为可交付的软件。

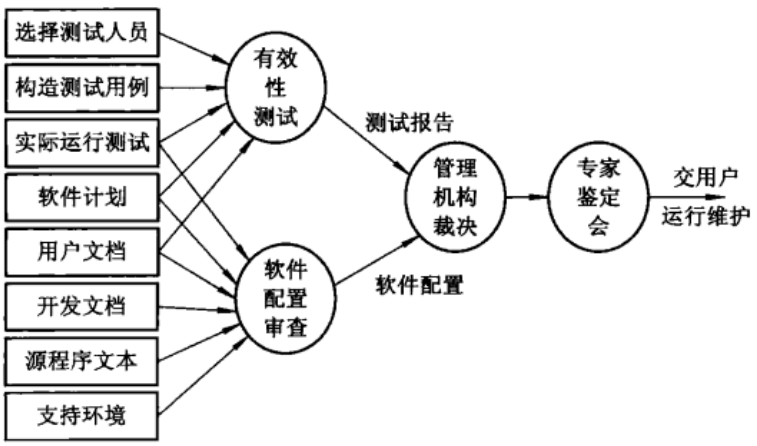


图 3.8　确认测试的步骤

1. 有效性测试

有效性测试是在模拟的环境下，运用黑盒测试的方法，验证被测软件是否满足需求规格说明书列出的需求。为此，需要制定测试计划，规定要做测试的种类，制定一组测试步骤，描述具体的测试用例。通过实施预定的测试计划和测试步骤，确定软件的特性是否与需求相符，确保所有的软件功能需求都能得到满足，所有的软件性能需求都能达到，所有的文档都是正确的且便于使用。

2. 软件配置复查

软件配置复查的目的是保证软件配置的所有成分，包括与实际运行环境中整个系统的支持环境都齐全，各方面的质量都符合要求。在确认测试的过程中，应当严格遵守用户手册和操作手册中规定的使用步骤，以便检查这些文档资料的完整性和正确性，记录发现的遗漏和错误，并且适当地补充和改正。

3.5 验 收 测 试

验收测试是以用户为主的测试，软件开发人员和质量保证人员也应参加。由用户参加

设计测试用例，通过用户界面输入测试数据，分析测试的输出结果。一般使用生产中的实际数据进行测试。在测试过程中，除了考虑软件的功能和性能外，还应对软件的可移植性、兼容性、可维护性、错误的恢复功能等进行确认。

3.5.1 α测试和β测试

在软件交付使用之后，用户将如何实际使用程序，对于开发者来说是无法预测的。因为用户在使用过程中常常会发生各种问题，如对软件操作使用方法的误解；异常的数据组合；产生对某些用户来说似乎是清晰的，但对另一些用户来说却难以理解的输出，等等。

α测试和β测试用于发现可能只有最终用户才能发现的错误。α测试可以是一个用户在开发环境下进行的测试，也可以是公司内部的用户在模拟实际操作环境下进行的测试。α测试是在受控制的环境下进行的测试。α 测试的目的是评价软件产品的 FURPS，即功能、可使用性、可靠性、性能和支持，尤其注重产品的界面和特色。α测试可以从软件产品编码结束之时开始，或在模块(子系统)测试完成之后开始，也可以在确认测试过程中产品达到一定的稳定度和可靠程度之后再开始。

β测试是由软件的多个用户在一个或多个用户的实际使用环境下进行的测试。与α测试不同的是，开发者通常不在测试现场。因而，β测试是在开发者无法控制的环境下进行的软件现场应用。在 β 测试中，由用户记下遇到的所有问题，包括真实的以及主观认定的，定期向开发者报告；开发者在综合用户的报告之后，做出修改，最后将软件产品交付给全体用户使用。β测试着重于产品的支持性，包括文档、客户培训和支持产品生产能力。只有当α测试达到一定的可靠程度时，才能开始β测试。

β测试具有如下一些意义：

(1) 广告效应。β版本本身已经很稳定了，在产品推出的形式上可能采取一些策略，以β版的形式推出，吸引一些好奇心强的用户去体验，效果可能要比直接推出要好。

(2) 调查市场。根据β测试统计的数据，分析用户的使用习惯，以便在下一个版本项目中改进。

(3) 查找 bug。β版本可以使用户协助测试，解决许多 bug 问题。

3.5.2 回归测试

软件生命周期中的任何一个阶段，只要软件发生了改变，就可能给该软件带来缺陷问题。而软件的改变可能是源于发现了错误并做了修改，也有可能是因为在集成或维护阶段加入了新的模块等多种情况。回归测试是一种验证已变更的系统的完整性与正确性的测试技术，是指重新执行已经做过的测试的某个子集，以保证修改没有引入新的错误或者发现由于更改而引起的之前未发现的错误，也就是保证改变没有带来非预期的副作用。因此，软件开发的各个阶段会进行多次回归测试。

1．回归测试的实施前提

(1) 当软件中所含错误被发现时，如果错误跟踪与管理系统不够完善，可能会遗漏对这些错误的修改；

(2) 开发者对错误理解得不够透彻，也可能导致所做的修改只修正了错误的外在表现，

而没有修复错误本身，从而造成修改失败；

(3) 修改还有可能产生副作用从而导致软件未被修改的部分产生新的问题，使本来工作正常的功能产生错误。

微软公司测试经验表明，一般修复3～4个错误会产生一个新的错误。同样，新代码加入软件的时候，除了新代码有可能含有错误外，还有可能给原有的代码带来影响。因此，软件一旦发生变化，必须重新补充新的测试用例，测试软件功能，确定修改是否达到预期目的，检查修改是否损害原有功能。

2. 回归测试的两个策略

回归测试是贯穿整个测试的各个阶段的测试活动，其目的是检验已经发现的缺陷有没有正确修改和修改过程中有没有引发新的缺陷。可以采用如下的策略进行回归测试。

1) 完全重复测试

完全重复测试是指将所有的测试用例全部再完全地执行一遍，以确认问题修改的正确性和修改后周边是否受到影响的测试方法。其缺点是由于要把用例全部执行，因此会增加项目成本，也会影响项目进度，所以很难完全执行。

2) 选择性重复测试

选择性重复测试是指可以选择一部分进行执行，以确认问题修改的正确性和修改后周边是否受到影响的测试方法。下面介绍几种选择性重复测试的方法：

(1) 覆盖修改法。覆盖修改法是指针对发生错误的模块，选取这个模块的全部用例进行测试。这种方法只能验证本模块是否还存在缺陷，不能保证周边与它有联系的模块不会因为这次改动而引发缺陷在修改范围内的测试。这类回归测试仅根据修改的内容来选择测试用例，仅保证修改的缺陷或新增的功能被实现，其效率最高，风险也最大，因为它无法保证这个修改是否影响了别的功能。该方法一般用于软件结构设计的耦合度较小的状态下。

(2) 周边影响法。周边影响法除了把出错模块的用例执行之外，把周边和它有联系的模块的用例也执行一遍，以保证回归测试的质量，需要分析修改可能影响到的那部分代码或功能，对于所有受影响的功能和代码，其对应的所有测试用例都将被回归。如何判断哪些功能或代码受影响，往往依赖于测试人员的经验和开发过程的规范性。

(3) 指标达成法。指标达成法根据一定的覆盖率指标选择回归测试。例如，规定修改范围内的测试阈值是90%，其它范围内的测试阈值为60%，该方法一般是在相关功能影响范围难以界定时使用。

(4) 基于操作剖面测试法。如果测试用例是基于软件操作剖面开发的，测试用例的分布情况将反映系统的实际使用情况。回归测试所使用的测试用例个数由测试预算确定，可以优先选择针对最重要或最频繁使用功能的测试用例，尽早发现对可靠性有最大影响的故障。

(5) 基于风险选择测试法。该方法根据缺陷的严重性来进行测试，基于一定的风险标准从测试用例库中选择回归测试包。选择最重要、关键以及可疑的测试，跳过那些次要的、例外的测试用例或功能相对稳定的模块。

3. 回归测试的流程

回归测试的流程一般具有如下步骤。

步骤1：在测试策略制定阶段，制定回归测试策略。

步骤 2：确定回归测试版本。

步骤 3：回归测试版本发布，按照回归测试策略执行回归测试。

步骤 4：回归测试通过，关闭缺陷跟踪单。

步骤 5：回归测试不通过，缺陷单返回；开发人员重新修改，再次做回归测试。

每当一个新的模块被当作集成测试的一部分加进来的时候，软件就发生了改变。新的数据流路径建立起来，新的 I/O 操作可能也会出现，还有可能激活新的控制逻辑。这些改变可能会使原本工作得很正常的功能产生错误。在集成测试策略的环境中，回归测试是对某些已经进行过的测试的某些子集再重新进行一遍，以保证改变不会传播无法预料的副作用。

4．回归测试与一般测试的比较

回归测试与一般测试相比具有许多不同点，下面分别从测试计划的可获性、测试范围、时间分配、开发信息、完成时间和执行效率等方面进行介绍。

(1) 测试计划的可获性：一般测试根据系统规格说明书和测试计划，测试用例都是新的；而回归测试可能是更改了的规格说明书、修改过的程序和需要更新的测试计划。

(2) 测试范围：一般测试的目标是检测整个程序的正确性；而回归测试的目标是检测被修改的相关部分的正确性以及它与系统原有功能的整合。

(3) 时间分配：一般测试所需时间通常是在软件开发之前预算；而回归测试所需的时间(尤其是修正性的回归测试)往往不包含在整个产品进度表中。

(4) 开发信息：一般测试关于开发的知识和信息可随时获取；而回归测试可能会在不同的地点和时间进行，需要保留开发信息以保证回归测试的正确。

(5) 完成时间：由于回归测试只需测试程序的一部分，因此完成测试所需时间通常比一般测试所需时间少。

(6) 执行频率：回归测试在一个系统的生命周期内往往要多次进行，一旦系统经过修改就需要进行回归测试。

3.6 思考与习题

一、选择题

1．软件测试是软件质量保证的重要手段，下述哪种测试是软件测试的最基础环节？(　　)

A. 功能测试　　B. 单元测试　　C. 结构测试　　D. 确认测试

2．确认测试计划是在软件生存周期的(　　)阶段制定的，在确认测试阶段完成的。

A. 需求分析　　B. 概要设计　　C. 综合测试　　D. 确认测试

3．单元测试的测试对象是(　　)。

A. 系统　　B. 程序模块　　C. 模块接口　　D. 系统功能

4．下列关于 α 测试的描述中正确的是(　　)。

A．α 测试需要用户代表参加　　B．α 测试不需要用户代表参加

C．α 测试是系统测试的一种　　D．α 测试是验收测试的一种

5．对于软件的 β 测试，下列哪些描述是正确的？(　　)

A. β测试就是在软件公司内部展开的，由专业测试人员执行的测试
B. β测试就是在软件公司内部展开的，由非专业测试人员执行的测试
C. β测试就是在软件公司外部展开的，由专业测试人员执行的测试
D. β测试就是在软件公司外部展开的，由非专业测试人员执行的测试

二、简答题解析

1. 软件测试的步骤是什么？各阶段关系如何？
2. 采用自顶向下的增值方式将模块按系统程序结构进行组装，其步骤是什么？
3. 单元测试有哪些内容？测试中采用什么方法?
4. α测试与β测试的区别是什么
5. 单元测试是什么？其主要任务是什么？
6. 集成测试方法有几种？集成测试与单元测试的区别是什么？
7. 常用的回归测试用例有哪几种选择方法？
8. 回归测试与一般测试有几点不同？分别是什么？

第4章 黑盒测试

本章详细地介绍黑盒测试方法，就等价类划分法、边界值分析法、决策表法、因果图法、场景法等测试方法进行详细的说明。

4.1 概 述

黑盒测试也称功能测试，它是通过测试来检测每个功能是否都能正常使用。在测试中，把程序看做一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，对程序接口进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息。

黑盒测试着眼于程序外部结构，不考虑内部逻辑结构，主要针对软件界面和软件功能进行测试。

黑盒测试是以用户的角度，从输入数据与输出数据的对应关系出发进行测试。黑盒测试注重于测试软件的功能需求，主要试图发现下列几类错误。

- 功能不正确或遗漏；
- 界面错误；
- 数据库访问错误；
- 性能错误；
- 初始化和终止错误等。

从理论上讲，黑盒测试只有采用穷举输入测试，把所有可能的输入都作为测试情况考虑，才能查出程序中所有的错误，下面分析一道例题。

【例 4-1】 任意输入三角形的三边，判断三角形类型。

如图 4.1 所示，输入三角形的三条边 a、b、c，设计测试用例数量。假设在字长为 16 位的计算机上运行，则每个整数可能的取值为 2^{16} 种，那么 a、b、c 三条边的各种可能取值的排列组合就有 $2^{16} \times 2^{16} \times 2^{16} \approx 3 \times 10^{14}$ 种，执行完所有的测试大约需要执行 3×10^{14} 次，也就是说，大约需要执行 3×10^{14} 次才能做到“穷尽”测试。假设执行 1 次需耗时 1 ms，则执行完所有的测试数据就需 1 万年。

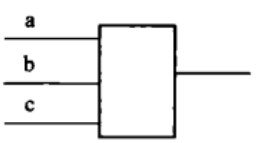


图 4.1 三角形三边取值情况的测试用例

因此，穷举测试是不可能的，所以要有针对性地选择测试用例。通过制定测试案例指导测试的实施，保证软件测试有组织、按步骤以及有计划地进行。具体的黑盒测试用例设计方法包括等价类划分法、边界值分析法、决策表法、因果图法、场景法等。

4.2 等价类划分法

等价类是指某个输入域的子集合。在该子集合中，测试某等价类的代表值就等于对这一类其它值的测试，对于揭露程序的错误是等效的。因此，全部输入数据合理划分为若干等价类，在每一个等价类中取一个数据作为测试的输入条件，就可以用少量代表性的测试数据取得较好的测试结果。

等价类划分为两种情况：有效等价类和无效等价类。

(1) 有效等价类：对于程序的规格说明来说是有意义的、合理的输入数据构成的集合，利用有效等价类可检验程序是否实现了规格说明中所规定的功能和性能。

(2) 无效等价类：与有效等价类相反，无效等价类是指对程序的规格说明无意义的、不合理的输入数据构成的集合。

4.2.1 划分原则

按照如下几条规则对等价类进行划分：

(1) 如果规定了输入值的范围，可定义一个有效等价类和两个无效等价类。

(2) 当规定了输入的规则时，则可以划分出一个有效的等价类(符合规则)和若干无效的等价类(从不同角度违反规则)。

(3) 如规定了输入数据的一组值，且程序对不同输入值做不同处理，则每个允许的输入值是一个有效等价类，并有一个无效等价类(所有不允许的输入值的集合)。

(4) 如规定了输入数据是整型，则可划分出正整数、零、负整数三个有效等价类。

(5) 当处理表格时，有效类可分为空表、含一项的表、含多项的表等。

4.2.2 设计测试用例的步骤

采用等价类设计测试用例一般经历如下几下步骤：

(1) 形成等价类表，每一等价类规定一个唯一的编号；

(2) 设计一测试用例，使其尽可能多地覆盖尚未覆盖的有效等价类，重复这一步骤，直到所有有效等价类均被测试用例所覆盖；

(3) 设计一新测试用例，使其只覆盖一个无效等价类，重复这一步骤直到所有无效等价类均被覆盖(通常程序执行一个错误后不继续检测其它错误，故每次只测试一个无效类)。

【例 4-2】 任意输入 3 个整数作为三角形的 3 条边的长度，判断三角形的类型。

【解答】 采用等价类划分法设计的测试用例如表 4.1 所示。

表 4.1 等价类划分法设计的测试用例

输入条件	有效等价类	无效等价类
是否能构成三角形的三条边	a>0,(1)	a<=0(7)
	b>0,(2)	b<=0(8)
	c>0,(3)	c<=0(9)
	a+b>c,(4)	a+b<=c (10)
	b+c>a,(5)	b+c<=a (11)
	a+c>b(6)	a+c<=b(12)
是否等腰三角形	a=b(13)	a!=b && b!=c && c!=a,(16)
	b=c(14)	
	c=a(15)	
是否等边三角形	a=b && b=c && c=a,(17)	a!=b(18)
		b!=c(19)
		c!=a(20)

设计测试用例输入数据：输入顺序是[a、b、c]。

序号	[a、b、c]	覆盖等价类	输 出
1	[3、4、5]	(1)、(2)、(3)、(4)、(5)、(6)	一般三角形
2	[0、1、2]	(7)	不能构成三角形
3	[1、0、2]	(8)	
4	[1、2、0]	(9)	
5	[1、2、3]	(10)	
6	[1、3、2]	(11)	
7	[3、2、1]	(12)	
8	[3、3、4]	(1)、(2)、(3)、(4)、(5)、(6)、(13)	等腰三角形
9	[3、4、4]	(1)、(2)、(3)、(4)、(5)、(6)、(14)	
10	[3、4、3]	(1)、(2)、(3)、(4)、(5)、(6)、(15)	
11	[3、4、5]	(1)、(2)、(3)、(4)、(5)、(6)、(16)	非等腰三角形
12	[3、3、3]	(1)、(2)、(3)、(4)、(5)、(6)、(17)	等边三角形
13	[3、4、4]	(1)、(2)、(3)、(4)、(5)、(6)、(14)、(18)	非等边三角形
14	[3、4、3]	(1)、(2)、(3)、(4)、(5)、(6)、(14)、(19)	
15	[3、3、4]	(1)、(2)、(3)、(4)、(5)、(6)、(14)、(20)	

4.3 边界值分析法

边界值分析法通过选择等价类边界设计测试用例。边界值分析法不仅重视输入条件边界，而且必须考虑输出域边界。它是对等价类划分方法的补充。

大量的错误是发生在输入或输出范围的边界上，而不是发生在输入或输出范围的内部，因此针对各种边界情况设计测试用例，可以查出更多的错误。

4.3.1 设计原则

使用边界值分析法设计测试用例，应确定输入和输出等价类的边界，应当选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据，而不是选取等价类中的典型值或任意值作为测试数据。

边界值分析方法具有如下原则：

(1) 如果输入条件规定了值的范围，则应选取刚达到范围的边界值，以及刚刚超越边界的值作为测试输入数据。

例如，输入变量为 X_1、X_2，取值范围是 $a \leqslant X_1 \leqslant b$，$c \leqslant X_2 \leqslant b$，边界分析图如图 4.2 所示。

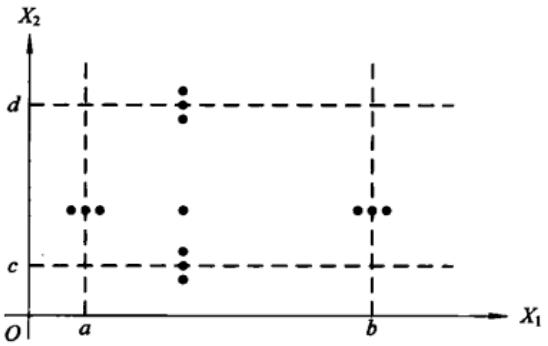


图 4.2 两变量函数边界分析测试用例

(2) 如果输入条件规定了值的个数，则用略低于最小值(Min−)、最小值(Min)、略高于最小值(Min+)、正常值(Normal)、略低于最大值(Max−)、最大值(Max)、略高于最大值(Max+)等值作为测试数据。

(3) 如果程序规格说明给出的输入域或输出域是有序集合，则应选取集合的第一个元素和最后一个元素作为测试用例。

4.3.2 应用举例

【例 4-3】 某报表处理系统要求用户输入处理报表的日期，日期限制在 2003 年 1 月至 2008 年 12 月，即系统只能对该期间内的报表进行处理，如日期不在此范围内，则显示输入错误信息。

【解答】采用边界值分析法设计的测试用例如表 4.2 所示。

表 4.2 边界值分析法设计的测试用例

测试用例编号	年	月	预期输出
Test1	2002	6	2002 年 6 月
Test2	2003	6	2003 年 6 月
Test3	2004	6	2004 年 6 月
Test4	2007	6	2007 年 6 月
Test5	2008	6	2008 年 6 月
Test6	2009	6	2009 年 6 月
Test7	2005	0	2005 年 0 月
Test8	2005	1	2005 年 1 月
Test9	2005	2	2005 年 2 月
Test10	2005	11	2005 年 11 月
Test11	2005	12	2005 年 12 月
Test12	2005	13	2005 年 13 月

4.4 决 策 表 法

决策表又称为判定表，是分析多种逻辑条件下执行不同操作的技术。在程序设计发展的初期，决策表作为程序编写的辅助工具。决策表可以把复杂的逻辑关系和多种条件组合情况表达明确，与高级程序设计语言中的 if-else、switch-case 等分支结构语句类似，将条件判断与执行的动作联系起来。但与程序语言中的控制语句不同的是，决策表能将多个独立的条件和多个动作联系清晰地表示出来。

决策表由四个部分组成，如图 4.3 所示。

图 4.3 决策表的组成

(1) 条件桩：列出了问题的所有条件，通常认为列出的条件次序无关紧要。

(2) 动作桩：列出了问题规定可能采取的操作，这些操作的排列顺序没有约束。

(3) 条件项：列出了针对其它条件桩的取值，在所有可能情况下的真假值。

(4) 动作项：列出了在条件项的各种取值的有机关联情况下应该采取的动作。

规则：任何条件组合的特定取值及其相应要执行的操作。在决策表中贯穿条件项和动作项的列就是规则。显然，决策表中列出多少条件取值，就有多少规则，条件项和动作项就有多少列。

所有条件都是逻辑结果(即真/假、是/否、0/1)的决策表称为有限条件决策表。如果条件有多个值，则对应的决策表叫做扩展条目决策表。使用决策表设计测试用例时，条件解释

为输入，动作解释为输出。

决策表适合以下特征的应用程序：

(1) if-then-else 分支逻辑突出；

(2) 输入变量之间存在逻辑关系；

(3) 涉及输入变量子集的计算；

(4) 输入和输出之间存在因果关系；

(5) 很高的圈复杂度。

决策表中具有 n 个条件的有限条目决策表有 2^n 个规则，可通过如下方法减少规则数目，如使用扩展条目决策表、代数简化表、查找条件条目的重复模式等。

如表 4.3 所示，打印机工作用决策表右上部分的“Y”表示它左边条件成立，“F”表示条件不成立，空白表示这个条件成立与否并不影响动作的选择。决策表右下部分中画“√”表示做它左边的相应动作，空白表示不做这项动作。

表 4.3 使用决策表设计打印机的测试用例

条件	不能打印	Y	Y	Y	Y	N	N	N
	红灯闪	Y	Y	N	N	Y	Y	N
	不能识别打印机	Y	N	Y	N	Y	N	Y
动作	检查电源线			√				
	检查打印机数据线	√		√				
	检查是否安装驱动程序	√		√		√		√
	检查墨盒	√	√			√	√	
	检查是否卡纸		√		√			

4.4.1 应用举例

使用判定表设计测试用例的具体步骤如下：

(1) 确定规则的个数。假如有 n 个条件，每个条件有两个取值(0，1)，则有 2^n 种规则。

(2) 列出所有的条件桩和动作桩。

(3) 填入条件项。

(4) 填入动作项。

(5) 简化，合并相似规则(相同动作)。

【例 4-4】 某国有企业改革重组，对职工重新分配工作的政策是：年龄在 20 岁以下者，初中文化程度脱产学习，高中文化程度当电工；年龄在 20 岁到 40 岁之间者，中学文化程度男性当钳工，女性当车工，大学文化程度都当技术员；年龄在 40 岁以上者，中学文化程度当材料员，大学文化程序当技术员。请用决策表描述上述问题的加工逻辑。

【解答】 条件取值表如表 4.4 所示。决策表如表 4.5 所示。

表 4.4 条件取值表

条 件 名	取 值	符 号	取 值 数
年龄	≤20	C	M1=3
	>20，<40	D	
	≥40	E	
文化程度	中学	G	M2=3
	高中	H	
	大学	I	
性别	男	M	M3=2
	女	F	

表 4.5 使用决策表分配工作的测试用例

	1	2	3	4	5	6	7	8	9	10
年龄	C	C	D	D	D	D	D	E	E	E
文化	G	H	H	G	G	H	I	G	H	I
性别	—	—	M	M	F	F	—	—	—	—
脱产学习	√									
电工		√								
钳工			√	√						
车工					√	√				
技术员							√			√
材料员								√	√	

4.4.2 优点和缺点

决策表把复杂问题的各种可能情况一一列出，易于理解。但是，决策表不能表达重复执行动作的缺点。B.Beizer 指出使用判定表设计测试用例的条件如下：

(1) 规格说明以判定表形式给出，或很容易转换成判定表。

(2) 条件的排列顺序不会也不影响执行哪些操作。

(3) 规则的排列顺序不会也不影响执行哪些操作。

(4) 每当某一规则的条件已经满足，并确定要执行的操作后，不必检验别的规则。

(5) 如果某一规则得到满足要执行多个操作，这些操作的执行顺序无关紧要。

这 5 个必要条件使得操作的执行完全依赖于条件的组合，对于不满足条件的判定表，可增加其它的测试用例。

4.5 因 果 图 法

等价类划分法和边界值分析法只是孤立地考虑各个输入数据的测试效果，没有考虑输入数据的组合及其相互制约关系。这样虽然各种输入条件可能出错的情况已经测试到了，但多个输入条件组合起来可能出错的情况却被忽视了。如果在测试时必须考虑输入条件的

各种组合，则可能的组合数目将是天文数字，因此必须考虑采用一种适合于描述多种条件的组合、相应产生多个动作的形式来进行测试用例的设计，这就需要利用因果图(逻辑模型)。

因果图利用图解法分析输入的各种组合情况，适合描述多种输入条件的组合、相应产生多个动作的方法。因果图法具有如下优点：

(1) 考虑多个输入之间的相互组合、相互制约关系。

(2) 指导测试用例的选择，指出需求规格说明描述中存在的问题。

(3) 能够帮助测试人员按照一定的步骤，高效率地开发测试用例。

(4) 因果图法是将自然语言规格说明转化成形式语言规格说明的一种严格的方法，可以指出规格说明存在的不完整性和二义性。

4.5.1 基本术语

下面介绍因果图的基本图形符号。

1) 原因—结果图

原因—结果图使用了简单的逻辑符号，以直线连接左右结点。左结点表示输入状态(原因)，右结点表示输出状态(结果)。图 4.4 表示规格说明中的 4 种因果关系，其中 c_i 表示原因，通常置于图的左部；e_i 表示结果，通常位于图的右部。c_i 和 e_i 均可取值 0 或 1(0 表示某状态不出现，1 表示某状态出现)。

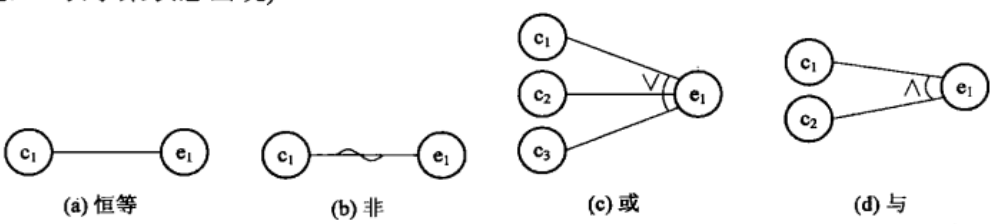


图 4.4 原因—结果图

图 4.4(a)表示“恒等”关系，即 c_1 是 1，则 e_1 是 1；c_1 是 0，则 e_1 为 0。图 4.4(b)表示“非”关系，即 c_1 是 1，则 e_1 是 0；c_1 是 0，则 e_1 是 1。图 4.4(c)表示“或”关系，“或”可有任意个输入。若 c_1、c_2 或 c_3 其中之一是 1，则 e_1 是 1；只有当 c_1、c_2、c_3 全为 0 时，e_1 才为 0。图 4.4(d)表示“与”关系，也可有任意个输入。若 c_1 和 c_2 都是 1，则 e_1 为 1；若 c_1 和 c_2 其中之一为 0，则 e_1 为 0。

2) 约束图

输入/输出状态相互之间存在的某些依赖关系，称为约束。例如，某些输入条件不可能同时出现等，如图 4.5 所示。

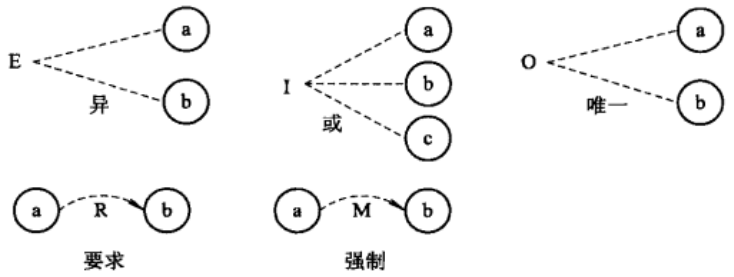


图 4.5 约束图

(1) E 约束(Exclusive，异)：a 和 b 中至多有一个可能为 1，即 a 和 b 不能同时为 1。

(2) I 约束(Inclusive，或)：a、b 和 c 中至少有一个是 1，即 a、b 和 c 不能同时为 0。

(3) O 约束(One and Only，唯一)：a 和 b 中必须有一个且仅有一个为 1。

(4) R 约束(Require，要求)：a 是 1 时，结果 b 是 1。

(5) M 约束(Masks，强制)：a 是 1 时，结果 b 是 0。

因果图法设计测试用例的步骤如图 4.6 所示。

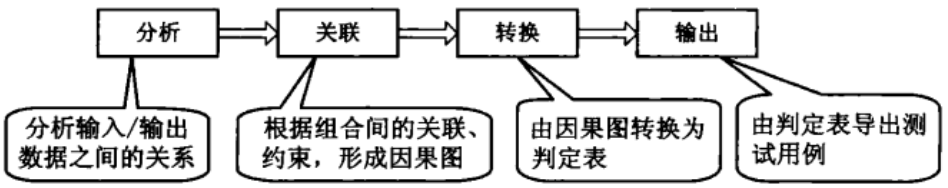


图 4.6 因果图法生成测试用例的步骤示意图

步骤 1：分析软件规格说明，哪些是原因(即输入条件或输入条件的等价类)，哪些是结果(即输出条件)，给每个原因和结果赋予标识符。

步骤 2：分析原因与结果之间、原因与原因之间对应的逻辑关系，用因果图的方式表示出来。

步骤 3：由于语法或环境限制，有些原因与原因之间、原因与结果之间的组合情况不可能出现，在因果图上用一些记号表明这些特殊情况的约束或限制条件。

步骤 4：把因果图转换为判定表。

步骤 5：从判定表的每一列产生出测试用例。

对于逻辑结构复杂的软件，先用因果图进行图形分析，再用判定表进行统计，最后设计测试用例。当然，对于比较简单的测试对象，可以忽略因果图，直接使用决策表。

4.5.2 应用举例

【例 4-5】 软件需求规格说明如下：第一列字符必须是 A 或 B，第二列字符必须是一个数字，在此情况下进行文件的修改。但如果第一列字符不正确，则给出信息 L；如果第二列字符不是数字，则给出信息 M。

【解答】 采用因果图方法，具体步骤如下：

步骤 1：分析程序规格说明书，识别哪些是原因，哪些是结果，原因往往是输入条件或者输入条件的等价类，而结果常常是输出条件。如下：

原因：

1—第一列字符是 A；

2—第一列字符是 B；

3—第二列字符是一数字。

结果：

21—修改文件；

22—给出信息 L；

23—给出信息 M。

步骤 2：根据原因和结果产生因果图，如图 4.7 所示。

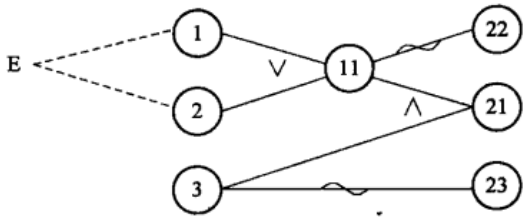


图 4.7 【例 4-5】因果图

步骤 3：状态 1 和状态 2 不能同时为 1，输入 3 个状态只有 6 种取值，如表 4.6 所示。

表 4.6 【例 4-5】的决策表

		1	2	3	4	5	6
原因	1	1	1	0	0	0	0
	2	0	0	1	1	0	0
	3	1	0	1	0	1	0
结果	21	1	0	1	0	0	0
	22	0	0	0	0	1	1
	23	0	1	0	1	0	1
测试用例		A3 A5	AM	B5 B4	BN B!	C2 X6	DY P;

4.6 场 景 法

软件流程的控制都是由事件触发的。同一事件不同的触发顺序和处理结果形成事件流，每个事件流触发时的情景便形成了场景。通过运用场景来对系统的功能点或业务流程进行描述，可以提高测试效果。场景法一般包含基本流和备选流，从一个流程开始，通过描述经过的路径来确定的过程，通过遍历所有的基本流和备选流来完成整个场景。

4.6.1 基本流和备选流

场景法的描述如图 4.8 所示，图中经过用例的每条路径都用基本流和备选流来表示。直黑线表示基本流，是经过用例的最简单的路径。备选流用不同的颜色表示，一个备选流可能从基本流开始，在某个特定条件下执行，然后重新加入基本流中(如备选流 1 和 3)；也可能起源于另一个备选流(如备选流 2)，或者终止用例而不再重新加入到某个流(如备选流 2 和 4)中。

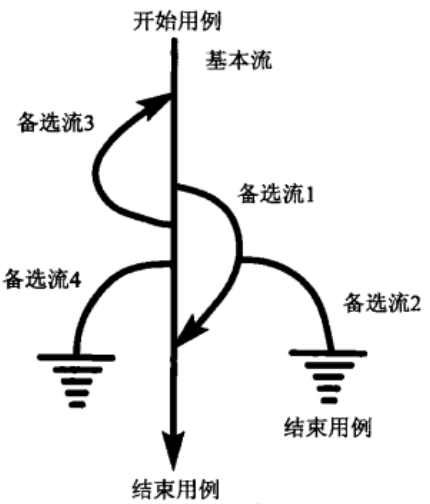


图 4.8 基本流和备选流

场景法的基本设计步骤如下所示：

步骤 1：根据说明，描述程序的基本流及各项备选流。

步骤 2：根据基本流和各项备选流生成不同的场景。

步骤 3：对每一个场景生成相应的测试用例。

步骤 4：对生成的所有测试用例重新复审，去掉多余的测试用例，测试用例确定后，对每一个测试用例确定测试数据值。

图 4.8 中，有一个基本流和四个备选流。每个经过用例的可能路径，确定不同的用例场景。从基本流开始，再将基本流和备选流结合起来，可以确定以下用例场景：

场景 1：基本流

场景 2：基本流->备选流 1

场景 3：基本流->备选流 1->备选流 2

场景 4：基本流->备选流 3

场景 5：基本流->备选流 3->备选流 1

场景 6：基本流->备选流 3->备选流 1->备选流 2

场景 7：基本流->备选流 4

场景 8：基本流->备选流 3->备选流 4

4.6.2 应用举例

【例 4-6】 采用场景法设计 ATM 系统的测试用例。

【解答】

1) 例子描述

图 4.9 是 ATM 系统的流程示意图。

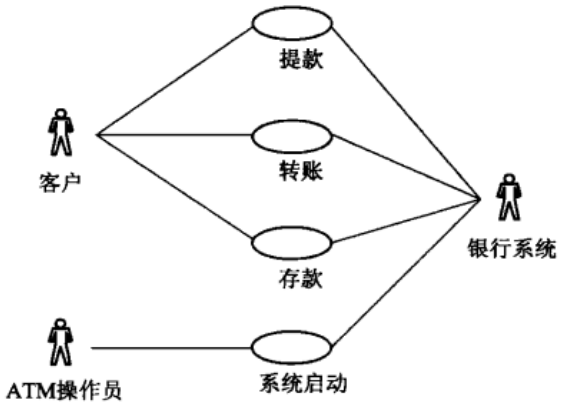


图 4.9 ATM 流程示意图

2) 场景设计

图 4.9 所示 ATM 系统中的基本流和某些备选流如表 4.7 所示。

表 4.7 基本流和备选流

基本流	本用例的开始是 ATM 处于准备就绪状态	
	步骤 1：	准备提款：客户将银行卡插入 ATM 机的读卡机
	步骤 2：	验证银行卡：ATM 机从银行卡的磁条中读取账户代码，并检查它是否属于可以接收的银行卡
	步骤 3：	输入 PIN(4 位)验证账户代码和 PIN，验证账户代码和 PIN，以确定该账户是否有效以及所输入的 PIN 对该账户来说是否正确。对于此事件流，账户是有效的，而且 PIN 对此账户来说正确无误。输入 PIN：ATM 要求客户
	步骤 4：	ATM 选项：ATM 显示在本机上可用的各种选项。在此事件流中，银行客户通常选择“提款”
	步骤 5：	输入金额：要从 ATM 中提取的金额。对于此事件流，客户需选择预设的金额(10 元、20 元、50 元或 100 元)。授权 ATM 通过将卡 ID、PIN、金额以及账户信息作为一笔交易发送给银行系统来启动验证过程。对于此事件流，银行系统处于联机状态，而且对授权请求给予答复，批准完成提款过程，并且据此更新账户余额
	步骤 6：	出钞：提供现金
	步骤 7：	返回银行卡：银行卡被返还
	步骤 8：	收据：打印收据并提供给客户。ATM 相应地更新内部记录
	用例结束时 ATM 又回到准备就绪状态	
备选流 1——银行卡无效	在基本流步骤 2 中，如果卡是无效的，则卡被退回，同时会通知相关消息	
备选流 2——ATM 内没有现金	在基本流步骤 4 中，如果 ATM 内没有现金，则“提款”选项不可用	
备选流 3——ATM 内现金不足	在基本流步骤 5 中，如果 ATM 机内金额少于请求提取的金额，则将显示一则适当的消息，并且在步骤 6 输入金额处重新加入基本流	
备选流 4——PIN 有误	在基本流步骤 3 中，客户有三次机会输入 PIN。如果 PIN 输入有误，ATM 将显示适当的消息；如果还存在输入机会，则此事件流在步骤 3 输入 PIN 处重新加入基本流。如果最后一次尝试输入的 PIN 码仍然错误，则该卡将被 ATM 机保留，同时 ATM 返回到准备就绪状态，本用例终止	
备选流 5——账户不存在	在基本流步骤 3 中，如果银行系统返回的代码表明找不到该账户或禁止从该账户中提款，则 ATM 显示适当的消息并且在步骤 8 返回银行卡处重新加入基本流	

续表

备选流 6——账面金额不足	在基本流步骤 6 中，银行系统返回代码表明账户余额少于在基本流步骤 5 输入金额内输入的金额，则 ATM 显示适当的消息并且在步骤 5 输入金额处重新加入基本流
备选流 7——达到每日最大的提款金额	在基本流步骤 6 中，银行系统返回的代码表明包括本次提款请求在内，客户已经或将超过在 24 小时内允许提取的最多金额，则 ATM 显示适当的消息并在步骤 5 输入金额上重新加入基本流
备选流 x——记录错误	如果在基本流步骤 9 中，记录无法更新，则 ATM 进入“安全模式”，在此模式下所有功能都将暂停使用，同时向银行系统发送一条适当的警报信息表明 ATM 已经暂停工作
备选流 y——退出	客户可随时决定终止交易(退出)。交易终止，银行卡随之退出
备选流 z——“翘起”	ATM 包含大量的传感器，用以监控各种功能，如电源检测器、不同的门和出入口处的测压器以及动作检测器等。在任一时刻，如果某个传感器被激活，则警报信号将发送给警方而且 ATM 进入“安全模式”。在此模式下所有功能都暂停使用，直到采取适当的重启/重新初始化的措施
第一次迭代中，根据迭代计划，我们需要核实提款用例已经正确地实施。此时尚未实施整个用例，只实施了下面的事件流： 基本流——提取预设金额(10 元、20 元、50 元、100 元) 备选流 2——ATM 内没有现金 备选流 3——ATM 内现金不足 备选流 4——PIN 有误 备选流 5——账户不存在/账户类型有误 备选流 6——账面金额不足	

表 4.8 所示是 ATM 生成的场景。

表 4.8 场 景 设 计

场景 1——成功提款	基本流	
场景 2——ATM 内没有现金	基本流	备选流 2
场景 3——ATM 内现金不足	基本流	备选流 3
场景 4——PIN 有误(还有输入机会)	基本流	备选流 4
场景 5——PIN 有误(不再有输入机会)	基本流	备选流 4
场景 6——账户不存在/账户类型有误	基本流	备选流 5
场景 7——账户余额不足	基本流	备选流 6

注：为方便起见，备选流 3 和 6(场景 3 和 7)内的循环以及循环组合未纳入上表。

3) 用例设计

对于这 7 个场景中的每一个场景都需要确定测试用例，一般采用矩阵或决策表来确定和管理测试用例。【例 4-6】中的测试用例包含测试用例 ID、场景/条件、测试用例中涉及的

所有数据元素和预期结果等项目。首先确定执行用例场景所需的数据元素，然后构建矩阵，最后要确定包含执行场景所需的适当条件的测试用例。表 4.9 中的“行”代表各个测试用例；“列”代表测试用例的信息；V 表示这个条件必须是有效的才可执行基本流；I 表示这种条件下将激活所需备选流；n/a 表示这个条件不适用于测试用例。

表 4.9 测试用例表

TC(测试用例)ID 号	场景/条件	PIN	账号	输入(或选择)的金额	账面金额	ATM 内的金额	预期结果
CW1	场景 1：成功提款	V	V	V	V	V	成功提款
CW2	场景 2：ATM 内没有现金	V	V	V	V	I	提款选项不可用，用例结束
CW3	场景 3：ATM 内现金不足	V	V	V	V	I	警告消息，返回基本流步骤 5，输入金额
CW4	场景 4：PIN 有误(还有不止一次输入机会)	I	V	n/a	V	V	警告消息，返回基本流步骤 3，输入 PIN
CW5	场景 4：PIN 有误(还有一次输入机会)	I	V	n/a	V	V	警告消息，返回基本流步骤 3，输入 PIN
CW6	场景 4：PIN 有误(不再有输入机会)	I	V	n/a	V	V	警告消息，卡予保留，用例结束

在表 4.9 中，六个测试用例执行了四个场景。测试用例 CW1 被称为正面测试用例，它一直沿着用例的基本流路径执行。基本流的全面测试必须包括负面测试用例，以确保只有在符合条件的情况下才执行基本流。这些负面测试用例由 CW2～CW6 表示。

每个场景只有一个正面测试用例和负面测试用例是不充分的，场景 4 正是这样的一个示例。要全面地测试场景 4(PIN 有误)，至少需要三个正面测试用例，以激活场景 4。

(1) 输入了错误的 PIN，但仍存在输入机会，此备选流重新加入基本流中的步骤 3(输入 PIN)。

(2) 输入了错误的 PIN，而且不再有输入机会，则此备选流将保留银行卡并终止用例。

(3) 最后一次输入了“正确”的 PIN。备选流在步骤 5(输入金额处)重新加入基本流。

注意：在表 4.9 中，无需为条件输入任何实际的值。以这种方式创建测试用例矩阵的一个优点在于容易看到测试的是什么条件。由于只需要查看 V 和 I，因此这种方式还易于判断是否已经确定了充足的测试用例。

4) 数据设计

一旦确定了所有的测试用例，则应对这些用例进行复审和验证以确保其准确且适度，并取消多余或等效的测试用例，如表 4.10 所示。

表 4.10 测试用例表

TC(测试用例)ID 号	场景/条件	PIN	账号	输入(或选择)的金额/元	账面金额/元	ATM 内的金额/元/	预期结果
CW1	场景 1：成功提款	4987	678-498	50.00	500.00	2000	成功提款。账户余额被更新为 450.00
CW2	场景 2：ATM 内没有现金	4987	678-498	100.00	500.00	0.00	提款选项不可用，用例结束
CW3	场景 3：ATM 内现金不足	4987	678-498	100.00	500.00	70.00	警告消息，返回基本流步骤 5，输入金额
CW4	场景 4：PIN 有误(还有不止一次输入机会)	4978	678-498	n/a	500.00	2000	警告消息，返回基本流步骤 3，输入 PIN
CW5	场景 4：PIN 有误(还有一次输入机会)	4978	678-498	n/a	500.00	2000	警告消息，返回基本流步骤 3，输入 PIN
CW6	场景 4：PIN 有误(不再有输入机会)	4978	678-498	n/a	500.00	2000	警告消息，卡予保留，用例结束

测试用例一经认可，就可以确定实际数据值并且设定测试数据。以上测试用例只是在本次迭代中需要用来验证提款用例的一部分测试用例。需要的其它测试用例包括以下内容:

- 场景 6——账户不存在/账户类型有误：未找到账户或账户不可用；
- 场景 6——账户不存在/账户类型有误：禁止从该账户中提款；
- 场景 7——账户余额不足：请求的金额超出账面金额。

4.7 思考与习题

一、选择题

1. 黑盒测试是通过软件的外部表现来发现软件缺陷和错误的测试方法，具体地说，黑盒测试用例设计技术包括(　　)等。

A. 等价类划分法、因果图法、边界值分析法、错误推测法、判定表驱动法

B. 等价类划分法、因果图法、边界值分析法、正交试验法、符号法

C. 等价类划分法、因果图法、边界值分析法、功能图法、基本路径法

D. 等价类划分法、因果图法、边界值分析法、静态质量度量法、场景法

2. 常用的黑盒测试方法有边界值分析、等价类划分、错误推测、因果图等。其中(　　)经常与其它方法结合起来使用。软件测试的步骤主要有单元测试、集成测试和确认测试。

A. 边界值分析　B. 等价类划分　C. 错误推测　D. 因果图

3. 等价类划分完成后，就可得出(　　)，它是确定测试用例的基础。

A. 有效等价类　B. 无效等价　C. 等价类表　D. 测试用例集

4. 在设计测试用例时，(　　)是用得最多的一种黑盒测试方法。

A. 等价类划分 B. 边界值分析 C. 因果图 D. 功能图

5. 在黑盒测试中，着重检查输入条件组合的测试用例设计方法是(　　)。

A. 等价类划分 B. 边界值分析 C. 错误推测法 D. 因果图

6. 除了测试程序外，黑盒测试还适用于对(　　)阶段的软件文档进行测试。

A. 编码 B. 软件详细设计 C. 软件总体设计 D. 需求分析

7. 由因果图转换出来的(　　)是确定测试用例的基础。

A. 判定表 B. 约束条件表 C. 输入状态表 D. 输出状态表

二、判断题

(1) 用黑盒法测试时，测试用例是根据程序内部逻辑设计的。(　　)

(2) 尽量用公共过程或子程序去代替重复的代码段。(　　)

(3) 测试是为了验证该软件是否已正确地实现了用户的要求。(　　)

(4) 尽量采用复合的条件测试，以避免嵌套的分支结构。(　　)

(5) 发现错误多的程序模块，残留在模块中的错误也多。(　　)

(6) 黑盒测试方法中最有效的是因果图法。(　　)

三、简答题

1. 等价类划分的原则是什么？

2. 边界值分析的设计原则是什么？有函数 $f(x, y, z)$，其中，$x \in [1900,2100]$，$y \in [1,12]$，$z \in [1,31]$，请写出该函数采用边界值分析法设计的测试用例。

3. 试采用等价类划分方法设计三角形类型的测试用例。

4. 试采用日期函数的决策表测试用例设计。

第5章 白盒测试

本章详细地介绍白盒测试方法。其中，逻辑覆盖法包括语句覆盖、判定覆盖、条件覆盖、条件判定覆盖、修正条件判定覆盖、条件组合覆盖和路径覆盖等测试方法，并介绍了基本路径测试方法的基本概念。

5.1 概 述

白盒测试是对软件的过程性细节做细致的检查，把测试对象看做一个打开的盒子，允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。通过在不同点检查程序状态，确定实际状态是否与预期的状态一致。

白盒测试只测试软件产品的内部结构和处理过程，而不测试软件产品的功能，用于纠正软件系统在描述、表示和规格上的错误，是进一步测试的前提。白盒测试分静态和动态两种：静态白盒测试是在不执行的条件下有条理地仔细审查软件设计、体系结构和代码，从而找出软件缺陷的过程，有时也称为结构分析。动态白盒测试也称结构化测试，通过查看并使用代码的内部结构，设计和执行测试。

5.2 逻辑覆盖法

白盒测试作为逻辑测试方法，是一种按照程序内部逻辑结构和编码结构设计测试用例的测试方法，目的是要测试程序中的判定和条件。测试程序逻辑结构通常需要通过使用控制流覆盖准则来定量度量测试的进行程度。

逻辑覆盖法又称为控制流覆盖，是选择一组实体以满足覆盖标准，如语句覆盖、判定覆盖、条件覆盖、条件判定覆盖、修正条件判定覆盖、条件组合覆盖和路径覆盖等，然后选择一组覆盖该组实体的有限路径。

下面通过例 5-1 具体讲解逻辑覆盖法的各种方法是如何设计测试用例的。

【例 5-1】 C++ 实现简单的数学运算。

【例 5-1】流程图如图 5.1 所示。其中Ⅰ、Ⅱ、Ⅲ、Ⅳ、Ⅴ是控制流上若干程序点。

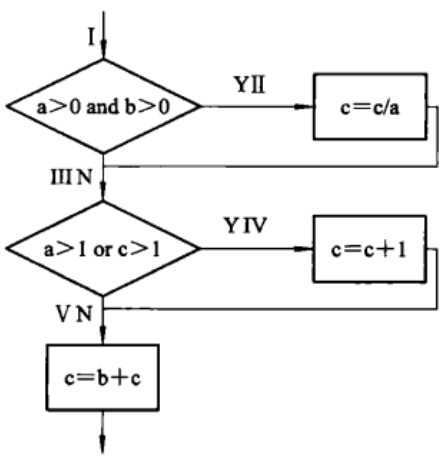


图 5.1 【例 5-1】程序流程图

5.2.1 语句覆盖

语句覆盖又称为线覆盖面或段覆盖面。其含义是指，选择足够数目的测试数据，使被测程序中每条语句至少执行一次。

如果例 5-1 测试用例选择 a=2，b=2，c=4，程序按照路径 I ->II->III->IV->V 执行，程序段中的 5 个语句均执行，符合语句覆盖。但是，如果测试用例选择 a=2，b=–2，c=4，程序按照路径 I ->III->IV->V 执行，则未能达到语句覆盖。

语句覆盖测试方法仅仅针对程序逻辑中的显式语句，对隐藏条件无法测试。如例 5-1 中第一个逻辑运算符“and”误写成“or”，则测试用例 a=2，b=2，c=4 仍能达到语句覆盖的要求，但是无法发现程序中的误写错误。

语句覆盖可以直接应用于目标代码，不需要处理源代码，但是作为最弱逻辑覆盖，语句覆盖对一些控制结构不敏感。由于逻辑覆盖很低，往往发现不了判断中逻辑运算符出现的错误。

5.2.2 判定覆盖

判定覆盖又称为分支覆盖或所有边覆盖，用于测试控制结构中布尔表达式为真或假(例如 if 语句和 while 语句)。布尔型表达式被认为是一个整体，取值为 true 或 false，而不考虑内部是否包含“逻辑与”或者“逻辑或”等操作符。

判定覆盖的基本思想是所设计的测试用例使得程序中每个判定分别取“真”分支和“假”分支并至少经历一次，即判断真假值均被满足。

例 5-1 判定覆盖测试用例如表 5.1 所示。

表 5.1 例 5-1 的判定覆盖测试用例

测试用例	a>0 and b>0	a>1 or c>1	执行路径
a=1，b=1，c=5	T	T	I ->II ->III->IV->V
a=1，b=–2，c=–3	F	F	I ->III->V
a=1，b=1，c=–3	T	F	I ->II ->III->V
a=1，b=–2，c=3	F	T	I ->III->IV->V

判定覆盖比语句覆盖具有更强的测试能力，多出了几乎一倍的测试路径。判定语句往往由多个逻辑条件组合而成(如判定语句中包含 and、or、case)，若仅仅判断其最终结果而忽略每个条件的取值情况，必然会遗漏部分测试路径。

由于短路操作符，判定覆盖忽略布尔表达式的内部分支。分析下面一段代码。

```
If(condition1 && condition2
    Statement1;
Else
    Statement2;
```

当 condition1 和 condition2 取值为真时，执行 Statement1 表达式；当 condition1 取值为假时，则 condition2 取值不进行判定，执行 Statement2 表达式。可以看到，这段代码控制结构的执行，短路操作符“&&”。表达式中第一个关系为假，则第二个就不进行判定。因此，

判定语句由多个逻辑条件组合而成，仅仅判断最终结果，忽略每个条件的取值情况，必然会遗漏部分测试路径。

5.2.3 条件覆盖

条件覆盖设计测试用例时，应使每个判断中每个条件的可能取值至少满足一次。

仍以例 5-1 为例，针对 a>0 and b>0 判定条件表达式，a>0 取值为“真”，记为 T1；a>0 取值为“假”，记为 F1；b>0 取值“真”，记为 T2；b>0 取值为“假”，记为 F2；条件表达式 a>1 or c>1，a>1 取值为“真”，记为 T3；a>1 取值为“假”，记为 F3；c>1 取值为“真”，记为 T4；c>1 取值为“假”，记为 F4。则条件覆盖测试用例如表 5.2 所示。

表 5.2 例 5-1 的条件覆盖测试用例

测试用例	覆盖条件	具体取值条件	执行路径
a=2，b=−1，c=−2	T1，F2，T3，F4	a>0，b<=0，a>1，c<=1	I ->III->IV->V
a=−1，b=2，c=3	F1，T2，F3，T4	a<=0，b>0，a<=1，c>1	I ->III->IV->V

条件覆盖比判定覆盖增加了对符合判定情况的测试，增加了测试路径。但是条件覆盖只能保证每个条件至少有一次为真，而不考虑所有的判定结果。表 5.2 中的测试用例 a=2，b=−1 和测试用例 a=−1，b=2 满足了条件覆盖的测试用例，保证了 a>0 and b>0 两个条件的可能值(True 和 False)至少满足一次，但是，由于测试用例的所有判定结果都是 False，并没有满足判定覆盖，因此条件覆盖不一定包含判定覆盖。

5.2.4 条件判定覆盖

既然判定条件不一定包含条件覆盖，条件覆盖也不一定包含判定覆盖，就自然会提出一种能同时满足两种覆盖标准的逻辑覆盖，这就是条件判定覆盖。

条件判定覆盖的含义是通过设计足够的测试用例，使得判断条件中的所有条件至少执行一次取值，同时，所有判断的可能结果至少执行一次。因此，条件判定覆盖的测试用例满足如下条件：

(1) 所有条件至少执行一次取值；

(2) 所有判断的可能结果至少执行一次。

例 5-1 的条件判定覆盖测试用例如表 5.3 所示。

表 5.3 例 5-1 的条件判定覆盖测试用例

测试用例	覆盖条件	执行路径
a=2，b=1，c=5	T1，T2，T3，T4	I ->II ->III->IV->V
a=−1，b=−2，c=−3	F1，F2，F3，F4	I ->III->V

条件判定覆盖能同时满足判定、条件两种覆盖标准，是判定和条件覆盖设计方法的交集，具有两者的简单性却没有两者的缺点。表面上看，条件判定覆盖测试了所有条件的取值，但事实并非如此，往往某些条件掩盖了另一些条件，并没有覆盖所有的“True 和 False”取值的条件组合情况，会遗漏某些条件取值错误的情况。为彻底地检查所有条件的取值，需要将判定语句中给出的复合条件表达式进行分解，形成由多个基本判定嵌套的流程图，

这样就可以有效地检查所有的条件是否正确了。

5.2.5 修正条件判定覆盖

修正条件判定覆盖是判定中每个条件的所有可能结果至少出现一次，每个判定本身的所有可能结果也至少出现一次，每个入口点和出口点至少要唤醒一次，并且每个条件都显示能单独影响判定结果。

MC/DC 定义的第一部分是标准的语句覆盖，第一和第二部分是条件判定覆盖准则，第四部分是 MC/DC 特有的判定条件。定义中最关键的字是“独立影响”，也就说明每一次每一个判定条件发生改变，必然会导致一次判定结果的改变，消除判定中的某些条件会被其它的条件所掩盖的问题，从而使得测试更加完备。

MC/DC 的目的就是消除测试过程中的各个单独条件之间的相互影响并且保证每个单独条件能够分别影响判定结果的正确性。

例如，A or B 的全部测试用例组合如表 5.4 所示。

表 5.4 A or B 的全部测试用例组合表

测试用例	A	B	结果
1	T	T	T
2	T	F	T
3	F	T	T
4	F	F	F

测试用例对(2，4)说明条件 A 独立地影响测试结果，测试用例对(3，4)说明条件 B 独立地影响测试结果，所以采用测试用例对(2，3，4)进行测试满足 MC/DC 覆盖准则。

MC/DC 继承了语句覆盖准则、条件判定覆盖准则、多重条件覆盖等判定条件，同时加入了新的判定条件。在条件判定覆盖准则中，不能够保证在模型中所有的条件都被覆盖，因为一个判定中的某些条件会被其它的一些条件所掩盖。如：任何一个条件与“1”进行“或”运算时，这个条件都不会起到任何的作用；同样，任何一个条件与“0”进行“与”运算时，这个条件都不会起到任何的作用。

例如，表 5.4 中 A or B 误写为 A and B，因为 $T \cap T = T \cup T$，且 $F \cap F = F \cup F$，所以两者所得到的判定结果相同，由此可说明虽然使用了判定条件覆盖准则来测试语句，逻辑表达式中的有些错误仍然不能检测出来。但如果使用 MC/DC 方法，就可以发现这样的错误，原因是 $T \cup F$ 值为 T，而 $T \cap F$ 值为 F，由此可说明中间的操作符号发生了错误。

MC/DC 具有如下优点：

(1) 继承了多重条件覆盖的优点；

(2) 线性地增加了测试用例的数量；

(3) 对操作数及非等式条件变化反应敏感；

(4) 具有更高的目标码覆盖率。

在许多软件系统中，尤其是以嵌入式和实时性为特征的航空机载软件中，MC/DC 得到了广泛的应用。如：MC/DC 已经被应用于 RTCA/DO-178B 标准当中，这个标准主要用于美国测试飞行软件的安全性审查。

5.2.6 条件组合覆盖

条件组合覆盖的基本思想：设计测试用例使得判断中每个条件的所有可能至少出现一次，并且每个判断本身的判定结果也至少出现一次，与条件覆盖的差别是条件组合覆盖不是简单地要求每个条件都出现“真”与“假”两种结果，而是要求这些结果的所有可能组合都至少出现一次。

条件组合覆盖是一种相当强的覆盖准则，可以有效地检查各种可能的条件取值的组合是否正确。它不但可覆盖所有条件的可能取值的组合，还可覆盖所有判断的可取分支，但仍可能会遗漏掉某些路径，测试还不完全。

例 5-1 的条件组合覆盖测试用例如表 5.5 所示。

表 5.5 例 5-1 的条件组合覆盖测试用例

编号	覆盖条件取值	判定条件取值	具体条件取值
1	T1，T2	表达式 a>0 and b>0 取 Y	a>0，b>0
2	T1，F2	表达式 a>0 and b>0 取 N	a>0，b<=0
3	F1，T2	表达式 a>0 and b>0 取 N	a<=0，b>0
4	F1，F2	表达式 a>0 and b>0 取 N	a<=0，b<=0
5	T3，T4	表达式 a>1 or c>1 取 Y	a>1，c>1
5	T3，F4	表达式 a>1 or c>1 取 Y	a>1，c<=1
7	F3，T4	表达式 a>1 or c>1 取 Y	a<=1，c>1
8	F3，F4	表达式 a>1 or c>1 取 N	a<=1，c<=1

表 5.5 合并后得到表 5.6

表 5.6 例 5-1 的条件组合覆盖测试用例

测试用例	覆盖条件	覆盖判断	覆盖组合	执行路径
a=2，b=1，c=5	T1，T2，T3，T4	表达式 a>0 and b>0 取 Y 分支 表达式 a>1 or c>1 取 Y 分支	编号 1， 编号 5	I ->II ->III->IV->V
a=2，b=−1，c=−2	T1，F2，T3，F4	表达式 a>0 and b>0 取 N 分支 表达式 a>1 or c>1 取 Y 分支	编号 2， 编号 5	I ->III->IV->V
a=−1，b=2，c=3	F1，T2，F3，T4	表达式 a>0 and b>0 取 N 分支 表达式 a>1 or c>1 取 Y 分支	编号 3， 编号 7	I ->III->IV->V
a=−1，b=−2，c=−3	F1，F2，F3，F4	表达式 a>0 and b>0 取 N 分支 表达式 a>1 or c>1 取 N 分支	编号 4， 编号 8	I ->III->V

条件组合覆盖准则满足判定覆盖、条件覆盖和条件判定覆盖准则，但线性地增加了测试用例的数量，并且条件覆盖组合不能保证所有的路径被执行测试。

5.2.7 路径覆盖

要想程序得到正确的运行结果，必须保证沿着特定的路径执行。路径覆盖的基本思想：选择足够的测试用例，使得程序中所有的可能路径都至少被执行一次。一条路径是从函数

的入口到出口分支的一个唯一序列。

例 5-1 的路径覆盖测试用例如表 5.7 所示。

表 5.7 例 5-1 的路径覆盖测试用例

测试用例	覆盖组合	执行路径
a=2，b=1，c=5	编号 1，编号 5	I ->II->III->IV->V
a=1，b=1，c=-3	编号 1，编号 8	I ->II->III->V
a=-1，b=2，c=3	编号 3，编号 7	I ->III->IV->V
a=-1，b=-2，c=-3	编号 4，编号 8	I ->III->V

路径覆盖比前面几种逻辑覆盖方法覆盖率都大，但也有如下缺点：随着程序代码复杂度的增加，测试工作量将呈指数级增长。例如：一个函数包含 10 个 if 语句，就有 $2^{10}=1024$ 条路径要测试，如果增加一个 if 语句，就有 $2^{11}=2048$ 条路径。二是许多路径由于数据相关不可能被执行。分析如下的代码段：

```
If(condition)
  Statement1;
  Statement2;
If(condition)
  Statement3;
  Statement4;
```

路径覆盖认为这个程序段包含 4 条路径，因为第一个 if 有两个分支，第二个 if 也有两个分支，组合为 4。但是，由于两个 if 的判定表达式完全相同，当 condition 为真，从代码入口到出口只有一条路径，所以实际上仅有 2 条路径而不是 4 条。

5.2.8 逻辑覆盖法总结

如图 5.2 所示，逻辑覆盖法中语句覆盖、判定覆盖、条件覆盖、条件判定覆盖、条件组合覆盖和路径覆盖具有相互包含的关系，其中语句覆盖最弱，从下至上依次增强，路径覆盖的效果最好。

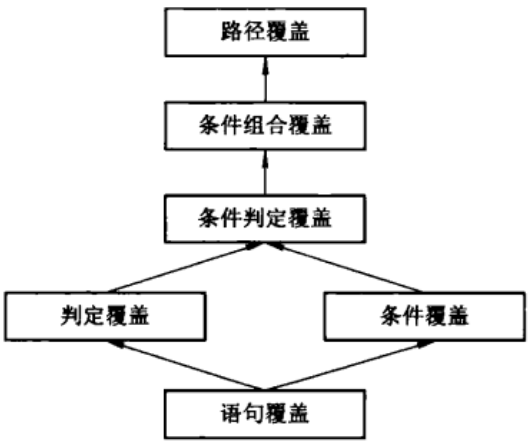


图 5.2 逻辑覆盖法总结

表 5.8 总结了逻辑覆盖各种方法的优缺点。根据测试用例设计的需要，逻辑覆盖法将不同的设计方法有效地结合起来，设计出覆盖率最大、最有效的测试用例。

表 5.8 逻辑覆盖法对比

方法	分支覆盖	条件覆盖	条件组合覆盖	路径覆盖
优点	简单，无须细分每个判定	增加了对符号判定情况的测试	对程序进行较彻底的测试，覆盖面广	清晰，测试用例有效
缺点	往往大部分的判定语句是由多个逻辑条件组合而成(如包含 and、or 等的组合)，若仅仅判断其组合条件的结果，而忽略每个条件的取值情况，必然会遗漏部分测试场景	达到条件覆盖，需要足够多的测试用例，但条件覆盖还是不能保证判定覆盖，这是由于 and 和 or 不同的组合效果造成的	对所有可能条件进行测试，需要设计大量、复杂测试用例，工作量比较大	类似于分支的方法，不能覆盖一些特定的条件，这些条件往往是容易出错的地方

逻辑覆盖法具有如下问题：语句或分支覆盖作为测试数据的主要依据，经过所选的路径并不能保证所有错误被查出，并且带有循环的程序将有无穷多条路径。

5.3 基本路径测试

基本路径测试给测试用例设计者提供方法来求出程序或过程设计中的逻辑复杂性测度，并使用该测度作为指南来定义执行路径的基本集，从该基本集导出的测试用例保证对程序中的每一条语句至少执行一次。

5.3.1 控制流

基本路径测试使用程序流程图来描述程序的结构性。程序流程图是一个有向图，又称为框图，采用不同图形符号标明条件或者处理等。由于这些符号在路径分析时不重要，为了突出控制流结构，将程序流程图进行简化，便产生控制流图。

控制流图是用于描述程序控制流的一种图示方法。控制流图只有节点和控制流线(或弧)两种图形符号，如图 5.3 所示。

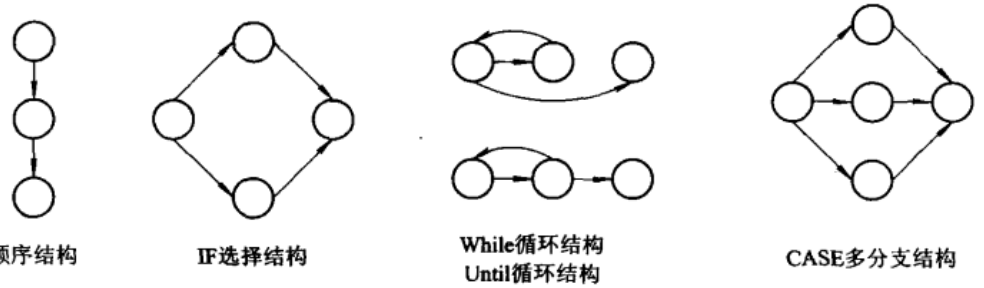


图 5.3 控制流图的基本符号示意图

(1) 节点：以标有编号的圆圈表示，用于表示程序流程图中矩形框、菱形框的功能，是一个或多个无分支的语句或源程序语句。

(2) 控制流线或弧：以箭头表示，与程序流程图中的流线功能一致，表示控制的顺序。

程序流程图简化成控制流图时，需注意以下情况：

(1) 在选择或多分支结构中，分支的汇聚处应有一个汇聚结点。

(2) 边和结点圈定的范围称为区域。需要注意，图形外的区域也应记为一个区域。

如图 5.4 所示，图(a)为程序流程图，图(b)表示图(a)转化的控制流图。

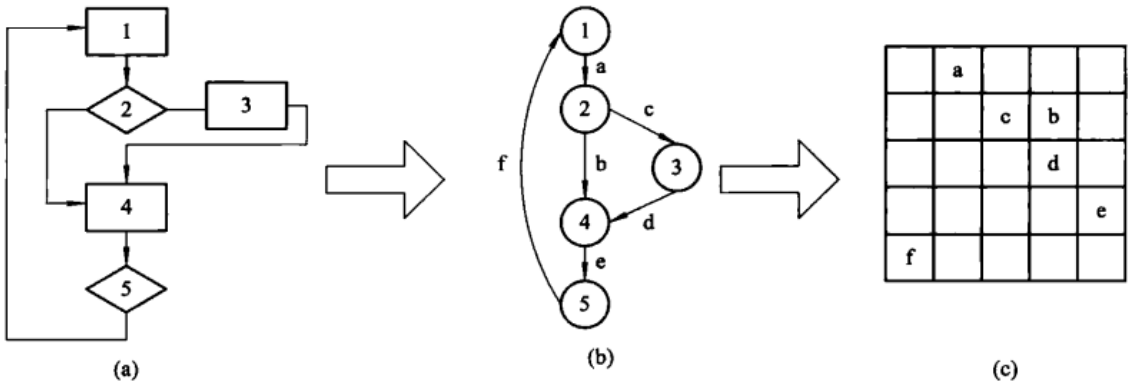


图 5.4 程序流程图转化为控制流图示意图

控制流图表示成矩阵的形式，称为控制流图矩阵。一个图形矩阵是一个方阵，其行列数目为控制流图中的结点数，行列依次对应到一个被标识的结点，矩阵元素对应到结点间的连接。控制流图的结点用数字标识，边用字母标识，第 i 结点到第 j 结点有 x 边相连接，则对应的图形矩阵中第 i 行与第 j 列有一个非空的元素 x。

图(c)表示图(b)的控制流图矩阵。控制流图的 5 个结点决定了图(c)矩阵共有 5 行 5 列。矩阵中 5 个元素 a、b、c、d、e 和 f 的位置对应所在控制流图中的号码。其中，弧 d 在图(b)中连接了结点 3 至结点 4，故在图(c)的矩阵中处于第 3 行第 4 列。需要注意控制流图的连接方向，图(b)中结点 4 到点 3 没有弧，则矩阵中第 4 行第 3 列没有元素。

为了评估程序的控制结构，控制流图矩阵项加入连接权值，连接权值为控制流提供了如下附加信息：

(1) 执行连接(边)的概率；

(2) 穿越连接的处理时间；

(3) 穿越连接时所需的内存；

(4) 穿越连接时所需的资源。

最简单情况是连接权值为 1(存在连接)或 0(不存在连接)。如图 5.5 所示，将图 5.4(c)的控制流图矩阵转化为连接矩阵。字母替换为 1，表示存在边(为表述清晰，0 没有画在图中)。

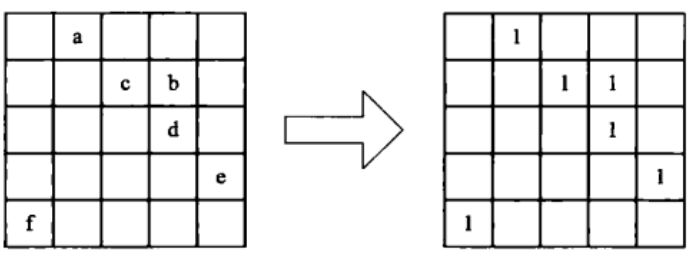


图 5.5 控制流图矩阵转化为连接矩阵

图 5.5 中含两个或两个以上项的行，表示判定节点，图 5.5 的判定节点为 1，这为计算环形复杂性提供了一种方法。

5.3.2 基本路径测试方法

基本路径是指所有程序路径作为一个集合，在这些路径当中必然存在一个最小路径的集合。基本路径测试通过确定测试用例是否完全覆盖基本路径而进行测试。

基本路径测试法是在程序控制流图的基础上，通过分析控制构造环路复杂性，导出基本可执行路径集合，设计测试用例。

基本路径测试法主要步骤如下所示。

步骤一：以详细设计或源代码作为基础，导出程序的控制流图。

步骤二：计算控制流图 G 的圈复杂度 V(G)。

圈复杂度为程序逻辑复杂性提供定量的测度，该度量用于计算程序的基本独立路径数目，确保所有语句至少执行一次的测试数量的上界。

步骤三：确定独立路径的集合，即确定线性无关的路径的基本集。

独立路径是指至少引入程序的一个新处理语句集合或一个新条件的路径，即独立路径必须包含一条在定义之前不曾使用的边。

步骤四：测试用例生成，确保基本路径集中每条路径的执行。

下面介绍三种计算控制流图的圈复杂度的方法。

方法一：控制流图 G 的圈复杂度 V(G)的定义为 V(G)=E−N+2，E 是图中边的数量，N 是图中结点的数量。

如图 5.6 所示，使用公式 V(G) = E − N + 2，其中 E 为 10，N 为 7，则 V(G) = 10 − 7 + 2 = 5，则圈复杂度为 5。

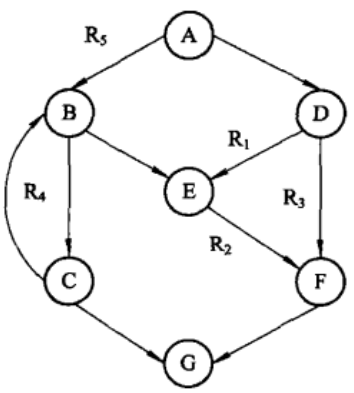


图 5.6 控制流图 G

方法二：将圈复杂度定义为控制流图中的区域数。

图 5.6 中的区域数为 5，故 V(G) = 5。

方法三：V(G) = P + 1，P 是控制流图 G 中判定(谓词)结点的数量。

图 5.6 中的判定结点为 A、B、C、D，即 P = 4，则根据公式，V(G) = P + 1 = 4 + 1 = 5。

方法四：将控制流图 5.6 转化为连接矩阵，根据图 5.5 原理，可得图 5.6 中的判定结点为 4，故圈复杂度同样为 5。

5.4 思考与习题

一、选择题

1. 以下不属于白盒测试技术的是(　　)。

A. 逻辑覆盖　　B. 基本路径测试　　C. 循环覆盖测试　　D. 等价类划分

2. 以下不属于逻辑覆盖的是(　　)。

A. 语句覆盖　　B. 判定覆盖　　C. 条件覆盖　　D. 基本路径

3. (　　)方法根据输出对输入的依赖关系设计测试用例。

A. 路径测试　　B. 等价类　　C. 因果图　　D. 归纳测试

4. McCabe 建议模块规模应满足 V(G)≤(　　)。

A. 20　　B. 10　　C. 30　　D. 40

二、简答题

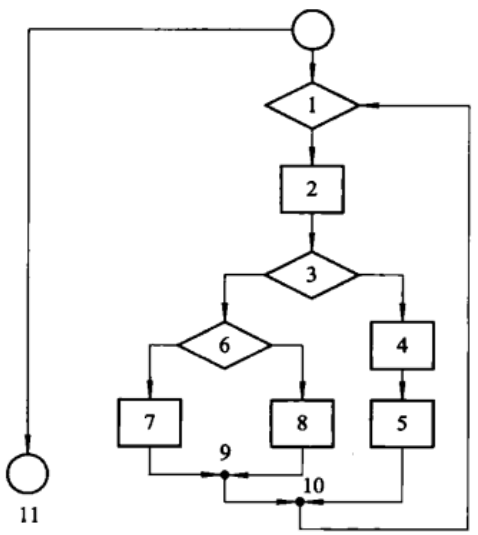
1. 白盒测试是什么？白盒测试和黑盒测试的区别体现在哪些方面？

2. 为什么说语句覆盖是最弱的逻辑覆盖？

3. 条件覆盖为什么不一定包含判定覆盖？

4. 采用白盒法进行测试时，测试用例覆盖路径的种类有哪几种？它们相互之间是什么关系？

5. 请把下面的程序流程图转换成控制流图。



第6章 自动测试技术

本章介绍软件自动测试的基本概念、自动测试发展历程、测试成熟度模型、自动化测试原理等内容。

6.1 自动测试技术简介

随着计算机日益广泛的应用，软件变得越来越庞大、越来越复杂，软件测试的工作量也随之增大。手工测试具有如下一些局限性：

(1) 通过手工测试无法做到覆盖所有代码路径；

(2) 许多与时序、死锁、资源冲突、多线程等有关的错误通过手工测试很难捕捉到；

(3) 在系统负载、性能测试，以及需要模拟大量数据或大量并发用户等各种应用场合时，很难通过手工测试进行；

(4) 在进行系统可靠性测试时，需要模拟系统运行十年、几十年，以验证系统能否稳定运行，手工测试无法模拟；

(5) 如果有大量(几千)的测试用例需要在短时间内完成，手工测试无法保证；

(6) 测试所有可能情况将遭遇“组合爆炸”问题，手工测试无法进行。

正因为手工测试存在如上的众多缺陷，自动测试技术应运而生。自动测试技术极大地减轻了手工测试的工作量，提高了软件测试的效率。

1. 自动测试的前提条件

自动测试是把以人为驱动的测试行为转化为机器执行的一种过程。实施自动测试之前需要对软件开发过程进行分析，以观察其是否适合使用自动测试。自动测试通常需要满足以下条件：

1) 软件需求变动不频繁

测试脚本的稳定性决定了自动测试的维护成本。如果软件需求变动过于频繁，测试人员需要根据变动的需求更新测试用例以及相关的测试脚本，较大的变动势必会增加维护脚本的成本，导致自动测试失败。一般情况下，对于需求分析中相对稳定的模块进行自动测试，而变动较大的模块仍采用手工测试。

2) 项目周期足够长

自动测试需求的确定、自动测试框架的设计、测试脚本的编写与调试均需要相当长的时间来完成。这样的过程本身就是一个测试软件的开发过程，需要较长的时间来完成。

3) 自动测试脚本可重复使用

在手工测试无法完成，需要投入大量时间成本与人力成本时，可考虑引入自动测试。

2. 自动测试过程

自动测试与软件开发过程从本质上来讲是一样的。自动测试过程可概括为利用自动化测试工具，经过对测试需求的分析，设计出自动测试用例，搭建自动测试的框架，设计与编写自动测试脚本，测试脚本的正确性，完成该套测试脚本。具体来讲可分为如下步骤：

1) 自动测试需求分析

当测试项目满足了自动化的前提条件，并确定在该项目中需要使用自动测试时，就可进行自动测试需求分析。此过程需要确定自动测试的范围以及相应的测试用例、测试数据，并形成详细的文档，以便于自动测试框架的建立。

2) 自动测试框架的搭建

自动测试框架与软件架构一样，定义了在使用该套脚本时需要调用哪些文件、结构，调用的过程，以及文件结构如何划分。搭建自动测试框架时需要将如下典型要素考虑进去。

(1) 公用的对象。不同的测试用例会有一些相同的对象被重复使用，比如窗口、按钮、页面等。这些公用的对象可被抽取出来，在编写脚本时调用。当这些对象的属性因为需求的变更而改变时，只需要修改该对象属性即可，无需修改所有相关的测试脚本。

(2) 公用的环境。测试用例使用相同的测试环境，将该测试环境独立封装，在各个测试用例中灵活调用，增强脚本的可维护性。

(3) 公用的方法。编写测试工具经常使用的方法，方便脚本的调用。

(4) 测试数据。一个测试用例往往需要执行多个测试数据，可将测试数据放在独立的文件中，由测试脚本执行到该用例时自动读取测试数据文件，以达到数据覆盖的目的。

3) 自动测试脚本的编写

自动测试脚本的编写过程是具体的测试用例的脚本转换。建议以录制为参考，以编写脚本为主要行为，避免录制脚本带来的冗余、公用元素的不可调用、脚本的调试复杂等问题。

4) 脚本的测试与试运行

事实上，测试用例形成的脚本通过测试后，并不意味着执行多个甚至所有的测试用例就不会出错。输入数据以及测试环境的改变，都会导致测试结果受到影响甚至失败。因此，脚本的测试与试运行极为重要，需要检查多个脚本不能按计划执行的原因，并保证其得到修复。

6.2　自动测试发展历程

如图 6.1 所示，自动测试发展大致经历了如下四个阶段。

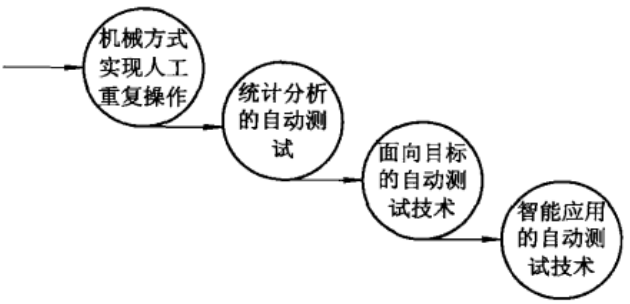


图 6.1　自动测试发展阶段

第一阶段：机械方式实现人工重复操作。

最初，自动测试主要研究如何采用自动方法来实现和替代人工测试中的繁琐和机械重复的工作。通过工人设计测试数据，对程序进行动态执行检测，脚本驱动自动执行。此时的自动测试活动只是软件测试过程的偶然行为，可在一定程度上提高效率，简化测试人员工作，但对整体的测试过程并无太大改进。

第二阶段：统计分析的自动测试。

只有自动测试结果具有可靠性，其使用才具有实际的意义。针对不同的测试准则和测试策略，指导测试的自动化过程以及对测试的结果进行评估。

第三阶段：面向目标的自动测试技术。

软件测试并不是机械和随机地发现错误，而是带有很强的目的性。进化计算和人工智能等技术，以及各种高性能的算法被引入自动测试技术。

第四阶段：智能应用的自动测试技术。

能力成熟度模型被引入软件工程，测试业界产生对应的测试成熟度模型。不同的自动测试等级成为测试好坏的一个衡量依据。

6.3 测试成熟度模型

测试成熟度模型(Testing capability Maturity Model，TMM)受 CMM 模型启发产生，主要关注测试成熟度模型。CMM 没有充分的定义测试，没有提及测试成熟度，没有对测试过程改进进行充分说明，在 KPA 中没有定义测试问题，与质量相关的测试问题，如可测性、充分测试标准、测试计划等方面也没有阐述。而 TMM 描述了测试过程，使得项目测试部分得到良好计划和控制。

TMM 测试成熟度可分解为如下 5 个级别：初始级、定义级、集成级、管理和测量级以及优化、预防缺陷和质量控制级。

1. 初始级

TMM 初始级软件测试过程的特点是测试过程无序，有时甚至是混乱的，几乎没有妥善定义的。初始级中软件的测试与调试常常被混为一谈，软件开发过程中缺乏测试资源、工具以及训练有素的测试人员。初始级的软件测试过程没有定义成熟度目标。

2. 定义级

TMM 的定义级中，测试已具备基本的测试技术和方法，软件的测试与调试已经明确地被区分开。这时，测试被定义为软件生命周期中的一个阶段，它紧随编码阶段之后，由于测试计划往往在编码之后才得以制订，这显然有背于软件工程的要求。

TMM 的定义级中需实现 3 个成熟度目标：制订测试与调试目标，启动测试计划过程，制度化基本的测试技术和方法。

1) 制订测试与调试目标

软件组织必须区分软件开发的测试过程与调试过程，识别各自的目标、任务和活动。正确区分这两个过程是提高软件组织测试能力的基础。与调试工作不同，测试工作是一种有计划的活动，可以进行管理和控制。这种管理和控制活动需要制订相应的策略和政策，以确定和协调这两个过程。

制订测试与调试目标包含5个子成熟度目标：

(1) 分别形成测试组织和调试组织，并有经费支持。

(2) 规划并记录测试目标。

(3) 规划并记录调试目标。

(4) 将测试和调试目标形成文档，并分发至项目涉及的所有管理人员和开发人员。

(5) 将测试目标反映在测试计划中。

2) 启动测试计划过程

测试计划作为过程可重复、可定义和可管理的基础，包括测试目的、风险分析、测试策略以及测试设计规格说明和测试用例。此外，测试计划还应说明如何分配测试资源，如何划分单元测试、集成测试、系统测试和验收测试。启动测试计划过程包含5个子目标：

(1) 建立组织内的测试计划组织并予以经费支持。

(2) 建立组织内的测试计划政策框架并予以管理上的支持。

(3) 开发测试计划模板并分发至项目的管理者和开发者。

(4) 建立一种机制，使用户需求成为测试计划的依据之一。

(5) 评价、推荐和获得基本的计划工具并从管理上支持工具的使用。

3) 制度化基本的测试技术和方法

改进测试过程能力，组织中应用基本的测试技术和方法，并说明何时和怎样使用这些技术、方法和支持工具。基本测试技术和方法制度化有如下2个子目标：

(1) 在组织范围内成立测试技术组，研究、评价和推荐基本的测试技术和测试方法，推荐支持这些技术与方法的基本工具。

(2) 制订管理方针以保证在全组织范围内一致使用所推荐的技术和方法。

3. 集成级

TMM的集成级中，测试不再是编码阶段之后的阶段，已被扩展成与软件生命周期融为一体的一组活动。测试活动遵循V字模型。测试人员在需求分析阶段便开始着手制订测试计划，根据用户需求建立测试目标和设计测试用例。软件测试组织提供测试技术培训，测试工具支持关键测试活动。但是，集成级没有正式的评审程序，没有建立质量过程和产品属性的测试度量。

集成级要实现如下4个成熟度目标：建立软件测试组织，制订技术培训计划，测试软件生命周期，控制和监视测试过程。

1) 建立软件测试组织

软件测试过程对软件产品质量有直接影响。测试往往是在时间紧、压力大的情况下完成一系列复杂活动，测试组完成与测试有关的活动，包括制订测试计划，实施测试执行，记录测试结果，制订与测试有关的标准和测试度量，建立测试数据库，测试重用，测试跟踪以及测试评价等。

建立软件测试组织要实现4个子目标：

(1) 建立全组织范围内的测试组，并得到上级管理层的领导和各方面的支持，包括经费支持。

(2) 定义测试组的作用和职责。

(3) 由训练有素的人员组成测试组。

(4) 建立与用户或客户的联系，收集他们对测试的需求和建议。

2) 制订技术培训计划

为高效率地完成好测试工作，测试人员必须经过适当的培训。

制订技术培训规划有 3 个子目标：

(1) 制订组织的培训计划，并在管理上提供包括经费在内的支持。

(2) 制订培训目标和具体的培训计划。

(3) 成立培训组，配备相应的工具、设备和教材。

3) 测试软件生命周期

提高测试成熟度和改善软件产品质量都要求将测试工作与软件生命周期中的各个阶段联系起来。该阶段有 4 个子目标：

(1) 将测试阶段划分为子阶段，并与软件生命周期的各阶段相联系。

(2) 基于已定义的测试子阶段，采用软件生命周期 V 字模型。

(3) 制订与测试相关的工作产品的标准。

(4) 建立测试人员与开发人员共同工作的机制。这种机制有利于将测试活动集成于软件生命周期中。

4) 控制和监视测试过程

软件组织采取如下措施：制订测试产品的标准，制订与测试相关的偶发事件的处理预案，确定测试里程碑，确定评估测试效率的度量，建立测试日志等。控制和监视测试过程有 3 个子目标：

(1) 制订控制和监视测试过程的机制和政策。

(2) 定义、记录并分配一组与测试过程相关的基本测量。

(3) 开发、记录并文档化一组纠偏措施和偶发事件处理预案，以备实际测试严重偏离计划时使用。

在 TMM 的定义级，测试过程中引入计划能力；在 TMM 的集成级，测试过程引入控制和监视活动。两者均为测试过程提供了可见性，为测试过程持续进行提供了保证。

4. 管理和测量级

TMM 的管理和测量级中，测试活动包括软件生命周期中各个阶段的评审、审查和追查，使得测试活动涵盖软件验证和确认活动。因为测试是可以量化并度量的过程，根据管理和测量级要求，与软件测试相关的活动，如测试计划、测试设计和测试步骤都要经过评审。为了测量测试过程，建立测试数据库，用于收集和记录测试用例，应记录缺陷并按缺陷的严重程度划分等级。

管理和测量级有 3 个要实现的成熟度目标：建立组织范围内的评审程序，建立测试过程的测量程序和软件质量评价。

1) 建立组织范围内的评审程序

软件组织应在软件生命周期的各阶段实施评审，以便尽早有效地识别、分类和消除软件中的缺陷。建立评审程序有 3 个子目标：

(1) 管理层要制订评审政策支持评审过程。

(2) 测试组和软件质量保证组要确定并文档化整个软件生命周期中的评审目标、评审计划、评审步骤以及评审记录机制。

(3) 评审项由上层组织指定。通过培训参加评审的人员，使他们理解和遵循相关的评审政策和评审步骤。

2) 建立测试过程的测量程序

测试过程的测量程序是评价测试过程质量、改进测试过程的基础，对监视和控制测试过程至关重要。测量包括测试进展、测试费用、软件错误和缺陷数据以及产品测量等。建立测试测量程序有3个子目标：

(1) 定义组织范围内的测试过程测量政策和目标。

(2) 制订测试过程测量计划。测量计划中应给出收集、分析和应用测量数据的方法。

(3) 应用测量结果制订测试过程改进计划。

3) 软件质量评价

软件质量评价内容包括定义可测量的软件质量属性，定义评价软件工作产品的质量目标等项工作。软件质量评价有2个子目标：

(1) 管理层、测试组和软件质量保证组要制订与质量有关的政策、质量目标和软件产品质量属性。

(2) 测试过程应是结构化、已测量和已评价的，以保证达到质量目标。

5. 优化、预防缺陷和质量控制级

本级的测试过程是可重复、可定义、可管理的，因此软件组织优化调整和持续改进测试过程。测试过程的管理为持续改进产品质量和过程质量提供指导，并提供必要的基础设施。

优化、预防缺陷和质量控制级有3个要实现的成熟度目标。

1) 应用过程数据预防缺陷

此时的软件组织能够记录软件缺陷，分析缺陷模式，识别错误根源，制订防止缺陷再次发生的计划，提供跟踪这种活动的办法，并将这些活动贯穿于全组织的各个项目中。应用过程数据预防缺陷的成熟度有如下4个子目标：

(1) 成立缺陷预防组。

(2) 识别和记录在软件生命周期各阶段引入的软件缺陷和消除的缺陷。

(3) 建立缺陷原因分析机制，确定缺陷原因。

(4) 管理、开发和测试人员互相配合制订缺陷预防计划，防止已识别的缺陷再次发生。缺陷预防计划要具有可跟踪性。

2) 支持软件质量控制

软件组织通过采用统计采样技术，测量组织的自信度，测量用户对组织的信赖度以及设定软件可靠性目标来推进测试过程。为了加强软件质量控制，测试组和质量保证组要有负责质量的人员参加，他们应掌握能减少软件缺陷和改进软件质量的技术和工具。支持统计质量控制的子目标有：

(1) 软件测试组和软件质量保证组建立软件产品的质量目标，如产品的缺陷密度、组织的自信度以及可信赖度等。

(2) 测试管理者要将这些质量目标纳入测试计划中。

(3) 培训测试组学习和使用统计学方法。

(4) 收集用户需求以建立使用模型。

3) 优化测试过程

优化测试过程在测试成熟度的最高级，已能够量化测试过程。这样就可以依据量化结果来调整测试过程，不断提高测试过程能力，并且软件组织具有支持这种能力持续增长的基础设施。基础设施包括政策、标准、培训、设备、工具以及组织结构等。优化测试过程包含：

(1) 识别需要改进的测试活动。

(2) 实施改进。

(3) 跟踪改进进程。

(4) 不断评估所采用的与测试相关的新工具和新方法。

(5) 支持技术更新。

优化测试过程所需子成熟度目标包括：

(1) 建立测试过程改进组，监视测试过程并识别其需要改进的部分。

(2) 建立适当的机制以评估改进测试过程能力和测试成熟度的新工具和新技术。

(3) 持续评估测试过程的有效性，确定测试终止准则。终止测试的准则要与质量目标相联系。

总之，TMM 的 5 个阶段可总结如下：

第一阶段：测试和调试没有区别，除了支持调试外，测试没有其它目的。

第二阶段：测试的目的是表明软件能够工作。

第三阶段：测试的目的是表明软件能够正常工作。

第四阶段：测试的目的不是要证明什么，而是要把软件不能正常工作的预知风险降低到能够接受的程度。

第五阶段：测试成为了自觉的约束，不用太多的测试投入即可产生低风险的软件。

综上所述，表 6.1 给出了测试成熟度模型的基本描述。

表 6.1 测试成熟度模型的基本描述

级别	简单描述	特 征	目 标
初始级	测试处于一个混乱的状态，缺乏成熟的测试目标，测试处于可有可无的地位	还不能把测试同调试分开；编码完成后才进行测试工作；测试的目的是表明程序没有错；缺乏相应的测试资源	
定义级	测试目标是验证软件是否符合需求，会采用基本的测试技术和方法	测试被看做是有计划的活动；测试同调试分开；在编码完成后才进行测试工作	启动测试计划过程；将基本的测试技术和方法制度化

续表

级别	简单描述	特　征	目　标
集成级	测试不再是编码后的一个阶段，而是贯穿在整个软件生命周期中，测试建立在满足用户或客户的需求上	具有独立的测试部门；根据用户需求设计测试用例；有测试工具辅助进行测试工作；没有建立起有效的评审制度；没有建立起质量控制和质量度量标准	建立软件测试组织；制订技术培训计划；测试在整个生命周期内进行；控制和监视测试过程
管理和度量级	测试是一个度量和质量的控制过程。在软件生命周期中，评审被作为测试和软件质量控制的一部分	进行可靠性、可用性和可维护性等方面的测试；采用数据库来管理测试用例；具有缺陷管理系统并划分缺陷的级别；还没有建立起缺陷预防机制，缺乏自动对测试中产生的数据进行收集和分析的手段	实施软件生命周期中各阶段评审；建立测试数据库并记录、收集有关测试数据；建立组织范围内的评审程序；建立测试过程的度量方法和程序；进行软件质量评价
优化级	具有缺陷预防和质量控制的能力，已经建立起测试规范和流程，并不断地进行测试改进	运用缺陷预防和质量控制措施；选择和评估测试工具存在一个既定的流程；测试自动化程度高；自动收集缺陷信息；有常规的缺陷分析机制	应用过程数据预防缺陷，统计质量控制，建立软件产品的质量目标，持续改进、优化测试过程

TMM 的特征可总结如下：

(1) TMM 从 CMM 延伸到测试，容易脱离测试过程的改进轨迹。例如，许多项目测试是从缺陷开始时，将缺陷划分为不同等级，先有测试用例，后有测试计划。

(2) 关注测试自身活动不够。TMM 中包括测试工具和测试数据库，但几乎没有涉及到缺陷清除率、自动化测试框架和测试流程等。

(3) 测试深度不够。测试具有很大风险因素，如何降低测试风险，提高测试用例的质量，TMM 并没有给出清晰的目标。

(4) 在过程监控、度量方面的内容较多，但是如何对实际测试能力(测试覆盖率、效率等)进行量化评估，TMM 并没有具体的对策，实施性较差。

6.4　三代测试框架

测试框架是为自动化软件测试提供支持的集合。测试框架经历了如下发展过程：

第一代，无测试框架。

无测试框架几乎没有起到自动化测试管理作用，测试需求与测试用例关联非常弱，对自动化测试人员的编程水平以及对测试方面的整体知识把握要求都非常高。

第二代，部分的测试框架。

部分的测试框架完成部分自动化测试管理，实现对测试用例的管理和缺陷跟踪，面向业务流，实现自动化测试需编写程序。

第三代，完整的测试框架。

完整的测试框架具有如下功能：

(1) 基于完整的 BPT 的测试框架；

(2) 拥有数据场景管理；

(3) 企业级的面向工作流的缺陷管理；

(4) 业务流复用框架，轻松完成；

(5) 高伸缩的自动执行框架。

6.5 自动测试原理

自动测试模拟人工对计算机的操作过程和操作行为，采用类似于编译的方法对程序代码进行检查。自动测试原理可概括如下：直接对代码进行静态和动态分析、测试过程的捕获和回放、测试脚本技术和虚拟用户技术。

1. 代码分析

代码分析是白盒测试的自动化方法，类似于高级编译系统，一般针对高级语言构造分析工具，定义类、对象、函数、变量等定义规则和语法规则，对代码进行语法扫描，找出不符合编码规范的地方，根据某种质量模型评价代码质量，生成系统的调用关系图等。

2. 录制回放

录制回放是黑盒测试的自动化方法，通过捕获用户的每一步操作，如用户界面的像素坐标或程序显示对象(窗口、按钮、滚动条等)的位置以及相应操作、状态变化或属性变化，用一种脚本语言记录描述，模拟用户操作。回放时，将脚本语言转换为屏幕操作，比较被测系统的输出记录与预先给定的标准结果。

目前的自动化负载测试解决方案几乎都是采用“录制—回放”的技术。所谓“录制—回放”技术，就是先由手工完成一遍测试流程，由计算机记录下这个流程期间客户端和服务器端之间的通信信息(这些信息通常是一些协议和数据)，并形成特定的脚本程序。然后在系统的统一管理下同时生成多个虚拟用户，并运行该脚本，监控硬件和软件平台的性能，提供分析报告或相关资料。通过模拟出成百上千的用户对应用系统进行负载能力的测试。

3. 脚本技术

1) 脚本概述

脚本是一组测试工具执行的指令集合，也是计算机程序的另一种表现形式。脚本至少具有如下的功能：

(1) 支持多种常用的变量和数据类型；

(2) 支持各种条件逻辑和循环结构；

(3) 支持函数的创建和调用。

脚本有两种：一种是手动编写或嵌入源代码；一种是通过测试工具提供的录制功能，运行程序自动录制生成脚本。录制生成脚本简单且智能化，容易操作，但仅靠自动录制脚本无法满足用户复杂的要求，因此需要手工添加函数进行参数设置，增强脚本的实用性。

手工编写脚本具有如下优点：

(1) 可读性好，流程清晰，检查点截取含义明确。

业务级的代码比协议级代码容易理解，也更容易维护，而录制生成的代码大多没有维护的价值。

(2) 手写脚本比录制脚本更真实地模拟应用。

录制脚本截获了网络包，生成协议级的代码，却往往忽略了客户端的处理逻辑，不能真实模拟应用程序的运行。

(3) 手写脚本比录制脚本更能提高测试人员的技术水平。

测试工具提供如 Java、VB、C 等高级程序设计语言的脚本，允许用户根据不同测试要求定义开发各种语言类型的测试脚本。

总之，使用哪种方式生成脚本，应以脚本模拟程序的真实有效性为准。例如，有些程序只需要执行多次迭代操作，没有特殊要求，选择自动生成的脚本即可。有些程序需要参数设置，则应使用手工脚本。

2) 脚本分类

脚本分为以下几种类型：

(1) 线性脚本。录制手工执行的测试用例得到的线性脚本，包含用户键盘和鼠标输入，用于检查某个窗口是否弹出等操作。

线性脚本具有如下一些优点：不需要深入的工作或计划，对实际执行操作可以审计跟踪。线性脚本适用于演示、培训或执行较少且环境变化小的测试、数据转换的操作功能。但是，线性脚本具有以下缺点：过程较繁琐，过多依赖于每次捕获的内容，测试输入和比较是“捆绑”在脚本中的，不能共享或重用脚本，容易受软件变化的影响。另外，线性脚本修改代价大，维护成本高，容易受意外事件影响，导致整个测试失败。

(2) 结构化脚本。结构化脚本类似于结构化程序设计，包含控制脚本执行指令，具有顺序、循环和分支等结构。结构化脚本的优点是健壮性好，通过循环和调用减少工作量，缺点是较复杂，而且测试数据仍然与脚本“捆绑”在一起。

(3) 共享脚本。共享脚本侧重描述脚本中共享的特性，脚本可以被多个测试用例使用，一个脚本可以被另一个脚本调用。当重复任务发生变化时，只需修改一个脚本，便可达到脚本共享的目的。

共享脚本具有如下优点：以较少的开销实现类似的测试，维护共享脚本的开销低于线性脚本。但是，共享脚本需要跟踪更多的脚本，给配置管理带来一定困难，并且对于每个测试用例仍然需要特定的测试脚本。

(4) 数据驱动脚本。数据驱动脚本将测试输入到独立的数据文件(数据库)中，而不是绑定在脚本中。执行时是从数据文件中读数据，使得同一个脚本执行不同的测试，只需对数据进行修改，不必修改执行脚本。通过一个测试脚本指定不同的测试数据文件，实现较多的测试用例，将数据文件单独列出，选择合适的数据格式和形式，达到简化数据、减少出错的目的。

数据驱动脚本具有如下优点：快速增加类似的测试用例，新增加的测试也不必掌握工具脚本技术，对以后类似的测试无需额外的维护，有利于测试脚本和输入数据分离，减少编程和维护的工作量，有利于测试用例的扩充和完善。但是，数据驱动脚本初始建立开销较大，需要专业人员支持。

(5) 关键字驱动脚本。关键字驱动作为比较复杂的数据驱动技术的逻辑扩展，是将数据文件变成测试用例的描述，用一系列关键字指定要执行的任务。关键字驱动技术假设测试者具有与被测系统相关的知识和技术，不必告之如何进行详细动作，以及测试用例如何执行，只说明测试用例即可。关键字驱动脚本多使用说明性方法和描述性方法。

4. 虚拟用户技术

虚拟用户技术通过模拟真实用户行为对被测程序(Application Under Test，AUT)施加负载，测量 AUT 的性能指标值，如事务的响应时间、服务器吞吐量等。虚拟用户技术以真实用户的“商务处理”(用户为完成一个商业业务而执行的一系列操作)作为负载的基本组成单位，用“虚拟用户”(模拟用户行为的测试脚本)模拟真实用户。

负载需求(例如并发虚拟用户数、处理的执行频率等)通过人工收集和分析系统使用信息来获得，负载测试工具模拟成千上万个来自不同 IP 地址、不同浏览器类型以及不同网络连接方式的虚拟用户同时访问 AUT，并实时监视系统性能，帮助测试人员分析测试结果。虚拟用户技术具有成熟测试工具支持，但确定负载的信息要依靠人工收集，因此准确性不高。

6.6 自动测试的 19 条经验教训

Elfriede Dustin 在总结了多年的自动测试项目经验后，提出了 19 条经验教训：

(1) 在软件开发周期中使用的各种工具不能够很轻易地整合在一起(The various tools used throughout the development lifecycle did not easily integrate)。

(2) 很多冗余的信息被存储在多个库中(Duplicate information was kept in multiple repositories)。

(3) 被测试工具牵着鼻子走(The automated testing tool drove the testing effort)。

(4) 整个测试组的每个人都在忙着编写自动测试脚本(Everyone on the testing staff was busy trying to automate scripts)。

(5) 重复开发的劳动，尝试编写一些非常复杂的测试脚本(Elaborate test scripts were developed，duplicating the development effort)。

(6) 自动测试脚本的创建往往会很麻烦，而不像工具厂商所吹嘘的那样简单易用(Automated test script creation was cumbersome)。

(7) 工具的培训开展得太迟，测试工程师缺乏工具方面的知识(Training was too late in the process，so test engineers lacked tool knowledge)。

(8) 测试工具在系统测试前两周才引入(The test tool was introduced to the testing program with two weeks left for system testing)。

(9) 测试人员对工具有抵触情绪(Testers resisted the tool)。

(10) 对自动化测试的期待值过高，期待及早得到回报(There were expectations of early payback)。

(11) 工具在识别第三方控件方面存在问题(The tool had problems recognizing third-party controls (widgets))。

(12) 缺乏测试脚本开发的规范性指南(A lack of test development guidelines was noted)。

(13) 某些测试工具需要插入代码到被测试程序中，但是开发人员直到测试后期才被告知这个问题(The tool was intrusive，but the development staff wasn't informed of this problem until late in the testing lifecycle)。

(14) 工具创建的报告没什么用处(Reports produced by the tool were useless)。

(15) 在尚未确定系统工程环境之前就选择和购买工具(Tools were selected and purchased before a system engineering environment was defined)。

(16) 工具的不同版本都在使用(Various tool versions were in use)。

(17) 工具的升级与现有的系统工程环境不兼容(The new tool upgrade wasn't compatible with the existing system engineering environment)。

(18) 工具的数据库不允许扩展(The tool's database didn't allow for scalability)。

(19) 未能正确地使用测试工具的管理功能，导致时间的浪费(Incorrect use of a test tool's management functionality results in wasted time)。

6.7　自动测试研究热点

目前，软件自动测试研究的主要热点如下。

1. 测试自动化框架

实际测试情况中，项目开发中同时包含了可实现自动化的测试活动以及难以完全实现自动化的测试活动。在自动测试与软件开发过程相融合的过程中，自动测试技术或测试方法难以完全匹配到整个项目中。而且，由于项目开发的动态特性，也难以使用通用的、普适的自动测试模型来完成测试，这就要求设计出具有较大可塑性的自动测试模型，在较少改动或者配置的情况下，最大化适应自动化测试的需求。

2. 测试自动化脚本技术

在测试过程中，大量的测试行为被自动化，测试人员指定一些简单的脚本，指导测试的自动化进行。例如，对应用程序的测试，人工的测试方式用脚本方式自动模拟执行，节省人力资源。但是，众多的自动测试工具包含了不同的脚本语言，测试人员必须学习不同的测试自动化脚本语言，如果测试脚本语言标准化且易于掌握，则可以简化自动测试的复杂性和减少测试人员的工作量。因此，对测试自动化脚本技术的研究也是自动测试技术中的一个主要内容。

3. 自动测试用例生成

最初的自动测试用例生成采用随机方式，这种方法简单且易于实现，但测试效率不高，受测试空间的分布影响，会有大量的测试冗余数据产生。采用启发式学习方法设计的测试用例在生成效率上有了较大的提高，包括使用进化计算或智能计算等技术，指导测试用例的自动生成，这些方法采用反馈方式对测试用例的产生进行自学习，可加速有效测试用例的生成，抑制和减少冗余数据的生成。

自动化测试用例的生成是一种典型的数据驱动测试技术，通过自动化测试数据的生成，测试程序中存在的缺陷或错误。由于测试用例对测试的重要性，以及测试数据自动构造实现上的可行性，使得这个领域成为自动测试技术中研究得最为广泛的技术之一。

4. 测试自动化中的预测

自动测试技术的发展不仅仅是自动化程度的提高，同时在测试的性能和效率上都提出了更高的要求。虽然测试的活动或行为已能够很大程度上实现自动化，但对测试结果的判定仍然依赖于人工干预。如果在测试过程中能通过特定的评测标准判断什么是程序错误，那么测试就能够实现真正意义的自动化。因此，测试预测是自动测试技术中非常关键的研究内容，直接关系到测试结果的准确性。

目前，对测试自动化中的预测问题的研究仍相对较少，研究大多集中在使用外部规则说明产生测试预测，该信息被作为自动测试过程发现错误的参照。

5. 自动测试与可靠性分析

自动测试是一个反复的过程，测试与可靠性之间的关系一直都是技术人员所关注的内容。究竟在多大程度上可以信任测试的结果，什么条件下所完成的测试才是充分的，只有解决了这些问题自动测试才能体现出其价值和意义。自动测试技术是为了提高测试的效率，各种自动化测试方法和技术的引入是否会影响测试的结果以及测试评测的可信度，这些都使得对自动测试的可靠性分析的研究非常有必要。

6. 自动化安全测试

该领域的研究主要集中在网络通信协议的安全性测试方面。随着网络应用的日益广泛，信息安全问题也日益严重，网络应用程序的安全性是测试的重要内容。针对网络安全的有限状态，自动测试机模型是该领域的主要研究内容。

6.8 思考与习题

简答题

1. 自动测试相对于手工测试有什么好处？
2. 自动测试发展经过了几个阶段？
3. 测试成熟度模型是什么？其包括哪些内容？
4. 录制和回放是指什么？
5. 脚本技术可以分为几类？分别是什么？

第7章 性能测试

本章介绍性能测试的基本概念，就负载测试、压力测试、可靠性测试、安全性测试、数据库测试、文档测试等给出详细说明，最后介绍了网站测试。

7.1 基本概念

性能测试在软件的质量保证中起着重要的作用，中国软件评测中心将性能测试概括为三个方面：应用在客户端性能的测试、应用在网络上性能的测试和应用在服务器端性能的测试。通常情况下，三方面有效、合理的结合，可以达到对系统性能全面的分析和对瓶颈的预测。

性能测试是指测试在一定条件下系统行为表现是否符合需求规格的性能指标。例如，测试传输的最长时限、传输的错误率、计算的精度、响应的时限和恢复时限等性能指标，其目的是验证软件系统是否能够达到需求规格说明中所提出的性能指标，同时发现软件系统中存在的性能瓶颈，优化软件系统。

性能测试主要包括以下几个方面：

(1) 评估系统的能力。测试中得到的负荷和响应时间数据被用于验证所计划的模型的能力，帮助作出决策。

(2) 识别体系中的弱点。通过将受控的负荷增加到一个极端的水平，确定体系的瓶颈或薄弱的地方，并进行修复。

(3) 系统调优。例如，通过长时间的测试执行将导致系统发生由于内存泄露而引起的失败，揭示系统中隐含的问题或冲突，进行调整，优化系统性能。

(4) 验证软件系统的稳定性和可靠性。

下面介绍一些常见的性能指标，如响应时间、并发用户数、吞吐量、性能计数器、休眠时间和点击率等。

1. 响应时间

响应时间是指“对请求做出响应所需要的时间”，可分解为网络传输时间、应用延迟时间、数据库延迟时间等。图 7.1 描述了 Web 应用的页面响应时间构成，页面的响应时间分解为“网络传输时间”(N1+N2+N3+N4)和“应用延迟时间”(A1+A2+A3)，而“应用延迟时间”又分为“数据库延迟时间”(A2)和“应用服务器延迟时间”(A1+A3)。

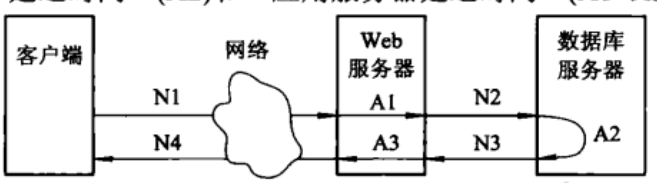


图 7.1 Web 应用的页面响应时间分解

一般而言，网站的响应时间有 2 秒、5 秒和 10 秒几个标准。2 秒之内响应客户被认为是“非常有吸引力的”，5 秒之内响应客户被认为是“比较不错的”，而 10 秒是客户能接受的响应时间的上限，也就是说打开网页花费时间超过了 10 秒钟，用户往往无法容忍。

2. 并发用户数

并发分为如下两种情况：一种是严格意义上的并发，即所有的用户在同一时刻做同一件事情或者操作，这种操作一般指做同一类型的业务。

另一种是广义范围的并发，与前一种并发的区别是，多个用户对系统发出了请求或者进行了操作，其请求或者操作可以是相同的，也可以是不同的。对整个系统而言，仍然是有多个用户同时对系统进行操作。后一种并发包含前一种并发，而且后一种并发更接近用户的实际使用情况。

实际的性能测试，测试人员比较关心业务并发用户数，也就是从业务角度关注应该设置多少个并发数比较合理。

下面给出估算并发用户数的公式。

$$C = \frac{nL}{T} \tag{1}$$

$$\hat{C} \approx C + 3\sqrt{C} \tag{2}$$

公式(1)中，C 是平均的并发用户数；n 是登录会话的数量；L 是登录会话的平均长度；T 指考察的时间段长度。

公式(2)给出了并发用户数峰值的计算方式，其中，$\hat{C}$ 指并发用户数的峰值，C 由公式(1)中得到，该公式是假设用户登录会话符合泊松分布而估算得到的。

【例 7-1】 一个软件系统每天大约有 400 个用户访问。用户在一天之内有 8 小时使用该系统，从登录到退出该系统的平均时间为 4 小时。

【解答】 根据公式(1)和公式(2)，得到：

$$C = 400 \times 4/8 = 200$$

$$\hat{C} \approx 200 + 3 \times \sqrt{200} = 242$$

为了较精确地计算并发用户数，下面给出一些建议。

(1) 时间粒度应较小。例如，设定 1 个小时为考察时间粒度，典型的 OA 系统将一天时间划分为 8 个区间，用于解决业务操作存在的时间集中性的问题。

(2) 考虑业务模式的特殊性。例如，员工打卡系统在上班前 30 分钟左右集中出现用户登录；账务系统在每月的结账日前比较繁忙；门户网站在发布重大消息前后会有访问高峰；旅游网站在节假日前夕会有大量用户访问等等。

3. 吞吐量

吞吐量是指在一次性能测试过程中网络上传输数据量的总和。一般来说，吞吐量用请求数每秒或页面数每秒来衡量。从业务角度分析，吞吐量用访问人数/天或者处理业务数/小时等衡量。其中，吞吐率 = 吞吐量/传输时间，体现软件性能承载能力。

吞吐量指标有如下两个作用：

(1) 协助设计性能测试场景，以及衡量性能测试场景是否达到了预期的设计目标。在设计性能测试场景时，吞吐量用于协助设计性能测试场景，通过估算吞吐量数据，测试场景

的事务发生频率等。

(2) 协助分析性能瓶颈。吞吐量是性能瓶颈的重要表现形式。因此，有针对性地测试吞吐量，可尽快定位到性能瓶颈所在位置。

吞吐量和并发用户数之间存在一定的联系。计算公式如下所示：

$$F = \frac{N_{vu} \times R}{T} \tag{3}$$

其中，F 表示吞吐量；N_{vu} 表示虚拟用户个数；R 表示每个虚拟用户发出的请求数量；T 表示性能测试所用的时间。遇到性能瓶颈状况，吞吐量和 VU 数量之间就不符合公式(3)了。图 7.2 为吞吐量—VU 数量关联图。

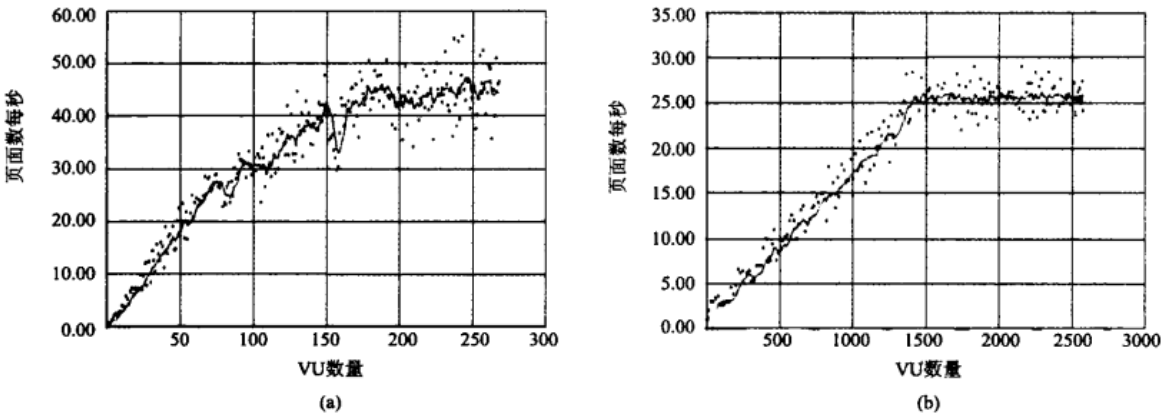


图 7.2 吞吐量—VU 数量关联图

如图 7.2 所示，吞吐量在 VU 数量增长到一定程度时产生性能瓶颈。对同一软件系统进行两次不同的性能测试，图 7.2(a)中 A 测试有 100 个并发，每个 VU 间隔 1 秒发出一个请求，测试吞吐量为 100 × 1/1 = 100；图 7.2(b)中 B 测试有 1000 个并发，每个 VU 间隔 10 秒发出一个请求，吞吐量为 1000 × 1/10 = 100，仍然是 100。但是执行测试 A 时，在 50 页每秒处出现性能瓶颈，而测试 B 在 25 页每秒处出现性能瓶颈。

4. 性能计数器

性能计数器是描述服务器或操作系统性能的一些数据指标，具有“监控和分析”作用。例如，Windows 系统的内存数、进程数、系统缓存等都是常见的性能计数器。与性能计数器相关的“资源利用率”，是指系统各种资源的使用状况。

$$\text{资源利用率} = \frac{\text{资源的实际使用}}{\text{总的资源可用量}}$$

例如，10000 个用户并发访问系统，Web 服务器 CPU 占用率为 68%，内存占用率为 55%，其中，68%和 55%就是资源利用率。

通过资源利用率进行横向对比，发现性能的瓶颈所在。例如，资源 A 的使用率达到了 100%，而其它资源利用率都较低，则资源 A 很有可能是系统的性能瓶颈所在。

通常的情况下，资源利用率需要结合响应时间变化曲线、系统负载曲线等各种指标进行综合的分析。

5. 休眠时间

休眠时间又称为思考时间，是指用户请求的间隔时间。在交互式应用中，用户不大可

能持续不断地发出请求，一般模式是用户发出一个请求，等待一段时间，再发出下一个请求。因此，自动化测试模拟用户操作就必须在测试脚本中让各个操作间隔一段时间，在操作语句之间设置 Think 函数，实现两个操作之间的等待时间。

休眠时间与迭代次数、并发用户数和吞吐量之间存在一定关系。公式(3)说明吞吐量是 VU 数量 N_{vu}、每个用户发出请求数 R 和时间 T 的函数。其中 R 可以用时间 T 和用户的思考时间 T_s 来计算，如下所示：

$$R = \frac{T}{T_s} \tag{4}$$

比较公式(3)和公式(4)可知，吞吐量与 N_{vu} 成正比，而与 T_s 成反比。在实际测试中，通过如下步骤计算休眠时间。

步骤 1：计算出系统的并发用户数；

步骤 2：统计出系统平均的吞吐量；

步骤 3：统计出平均每个用户发出的请求数量；

步骤 4：根据公式(4)计算出思考时间。

当然，为了让性能测试场景更加符合实际情况，以计算思考时间为基准，在一定幅度内随机变动。

6. 点击率

点击率是指每秒用户向 Web 服务器提交的 HTTP 请求数，作为 Web 应用的特有指标。Web 应用是“请求—响应”模式，即用户发出一次请求，服务器响应请求，处理后返回给用户。点击是 Web 应用能够处理的最小单位，点击率越大，服务器的压力也就越大。需要注意的是，这里的点击并非指鼠标的一次单击操作，因为在一次单击操作中，客户端可能向服务器发出多个 HTTP 请求。

7.2 性能测试分类

下面介绍一些常见的性能测试。

7.2.1 负载测试

1. 概述

负载测试(Load Test)是通过测试系统在资源超负荷情况下的表现，以发现设计上的错误或验证系统的负载能力。在这种测试中，将使测试对象承担不同的工作量，以评测和评估测试对象在不同工作量条件下的性能行为，以及持续正常运行的能力。负载测试的目标是确定并确保系统在超出最大预期工作量的情况下仍能正常运行。此外，负载测试还要评估性能特征，例如响应时间、事务处理速率和其它与时间相关的方面。

负载测试是模拟实际软件系统所承受的负载条件的系统负荷，通过不断加载(如大量重复的行为、逐渐增加模拟用户的数量)或其它加载方式来观察不同负载下系统的响应时间和数据吞吐量、系统占用的资源(如 CPU、内存)等，以检验系统的行为和特性，以发现系统可

能存在的性能瓶颈、内存泄漏、不能实时同步等问题。

2. 测试的加载方式

负载测试的加载方式通常有如下几种：

(1) 一次加载。一次性加载某个数量的用户，在预定的时间段内持续运行。例如，早晨上班时间，访问网站或登录网站的时间非常集中，基本属于扁平负载模式。

(2) 递增加载。有规律地逐渐增加用户，每几秒增加一些新用户，交错上升。借助这种负载方式的测试，容易发现性能的拐点，即性能瓶颈。

(3) 高低突变加载。某个时间用户数量很大，突然降到很低，过一段时间，又突然加到很高，反复几次。借助这种负载方式的测试，容易发现资源释放、内存泄露等问题。

(4) 随机加载方式。由随机算法自动生成某个数量范围内变化的、动态的负载，这种方式可能是和实际情况最为接近的一种负载方式。虽然不容易模拟系统运行出现的瞬时高峰期，但可以模拟系统长时间的高位运行状态。

7.2.2 压力测试

1. 概述

压力测试(Stress Test)也称强度测试，是在强负载(大数据量、大量并发用户等)下的测试，通过查看应用系统在峰值使用情况下的操作行为，发现系统的某项功能隐患、系统是否具有良好的容错能力和可恢复能力。压力测试分为高负载下的长时间(如 24 小时以上)的稳定性压力测试和极限负载情况下导致系统崩溃的破坏性压力测试。

压力测试可以看做是负载测试的一种，即高负载下的负载测试，或者说压力测试采用负载测试技术。通过压力测试，可以发现内存泄漏问题，还可以发现影响系统稳定性的问题。例如，在正常负载情况下，某些功能不能正常使用或系统出错的概率比较低，可能一个月只出现一次，但在高负载(压力测试)下，可能一天就出现，从而发现有缺陷的功能或其它系统问题。

微软测试实践经验表明，如果软件产品通过了 72 小时压力测试，则在 72 小时后出现问题的可能性微乎其微。所以，72 小时成为微软产品压力测试的时间标志。压力测试用例的参考模板如图 7.3 所示。

1. 被测试对象的介绍
2. 测试范围与目的
3. 测试环境与测试辅助工具的描述
4. 测试驱动程序的设计
5. 压力测试用例

极限名称A	如“最大并发用户数量”	
前提条件		
输入/动作	输出/响应	是否能正常运行
如10个用户并发操作		
如20个用户并发操作		

⋮

图 7.3 压力测试用例的参考模板

2. 测试步骤

压力测试检查系统在资源超负荷的情况下的表现，压力测试的其中一个变种——敏感测试是指在有些情况下，数据界限内的很小范围的数据可能会引起错误的运行，或引起性能急剧下降。敏感测试用于发现可能会引起不稳定或错误处理的数据组合。

压力测试的一般步骤如下：

步骤一：进行简单多任务测试。

步骤二：简单压力缺陷修正后，增加系统的压力直到系统崩溃。

因此，压力测试的主要目的是度量应用系统的性能和扩展性。在实施并发负载过程中，通过实时性能监测来确认和查找问题，并针对所发现问题对系统性能进行优化。压力测试工具能够对整个企业架构进行测试，通过这些测试，企业能最大限度地缩短测试时间，优化性能和加速应用系统的发布周期。

3. 性能测试、负载测试和压力测试的区分

性能测试、负载测试和压力测试常常容易混淆，难以区分，从而造成不正确的理解和错误的使用。由于负载测试、压力测试和性能测试往往在测试手段和方法上比较相似，通常会使用相同的测试环境和测试工具，而且都会监控系统所占用资源的情况以及其它相应的性能指标，因此容易产生概念混淆。

性能测试、负载测试和压力测试三者的测试目的是不同的。性能测试是为了获得系统在某种特定的条件下(包括特定的负载条件下)的性能指标数据；而负载测试、压力测试是为了发现软件系统中所存在的问题，包括性能瓶颈、内存泄漏等。

当通过负载测试为了获得系统正常工作时所能承受的最大负载时，负载测试就成为容量测试。压力测试用于测试系统在何种极限情况下会崩溃、系统是否具有自我恢复性等，但更多的是为了确定系统的稳定性。

负载测试与压力测试区分如下：负载测试是通过逐步增加系统复杂性来测试其变化，看最后在满足性能的情况下，系统最多能接受多大的负载的测试。压力测试是通过逐步加系统复杂性来测试其变化，看最后在满足性能的情况下，施加多大压力能使系统处于失效的状态。通俗来说，就是发现系统在什么条件下其性能会变得不可接受。压力测试是一种特定类型的负载测试。

负载测试与性能测试区分如下：负载测试是为了发现系统的性能问题。负载测试需要通过系统性能特性或行为来发现问题，从而为性能改进提供帮助。从这个意义看，负载测试可以看做性能测试的一部分。但它们两者的目的是不一样的，负载测试是为了发现缺陷，只测试在一些极端条件下，系统还能否正常工作，或加载到系统崩溃而找出系统性能的瓶颈。而性能测试是为了获取性能指标，因为在性能测试过程中，也可以不调整负载，而是在同样负载情况下改变系统的结构、算法、硬件配置等来得到性能指标数据。从这个意义看，负载测试可以看做是性能测试的一种技术，即性能测试使用负载测试的技术和工具。性能测试要获得在不同的负载情况下的性能指标数据。

7.2.3 可靠性测试

软件可靠性是软件质量的一个重要标志。美国电气和电子工程师协会(IEEE)将软件可靠

性定义为：系统在特定的环境下，在给定的时间内无故障地运行的概率。软件可靠性涉及软件的性能、功能性、可用性、可服务性、可安装性、可维护性等多方面特性，是对软件在设计、生产以及在它所预定环境中具有所需功能的置信度的一个度量。

可靠性测试一般伴随着强壮性测试，是评估软件在运行时的可靠性。通过可靠性测试，确认平均无故障时间(MTTF，Mean Time To Failure)、故障发生前平均工作时间(MTTFF，Mean Time TO First Failure)或因故障而停机的时间(MTTR，Mean Time To Repairs)在一年中应不超过多少时间。可靠性测试强调随机输入，并通过模拟系统实现，很难通过实际系统的运行来实现。

7.2.4 数据库测试

数据库测试一般包括数据库的完整测试和数据库容量测试。下面依次介绍。

1) 数据库完整测试

数据库完整测试是指测试关系型数据库完整性原则以及数据合理性测试。数据库完整性原则是指：

(1) 主码完整性：主码不能为空。

(2) 外码完整性：外码必须等于对应的主码或者为空。

(3) 用户自定义完整性。例如性别字段的取值只能是“男”或者“女”。

比如，某MIS系统有两张表：部门表和员工表。其中，部门表有部门编号、部门名称、部门经理等字段，主码为部门编号；员工表中有员工编号、员工所属部门编号、员工名称、员工类型等字段，主码为员工编号，外码为员工所属部门编号，对应部门表。如果某条部门记录中的部门编号或员工记录中的员工编号为空，就违反了“主码完整性”原则。如果某个员工所属部门的编号为##，但是##在部门编号中却找不到，就违反了“外码完整性”原则。

又例如，通过用户自定义完整性将员工表的年龄属性限制在20～35岁之间，如果用户输入的年龄不在这个范围之内，就违反了“用户自定义完整性”原则。

2) 数据库容量测试

数据库容量测试指通过存储过程往数据库表中插入一定数量的数据，看相关页面是否能够及时显示数据。数据库容量测试使测试对象处理大量的数据，以确定是否达到了使软件发生故障的极限。数据库容量测试还将确定测试对象在给定时间内能够持续处理的最大负载或工作量。比如，通过insert customer往员工表中插入10000个数据，看其是否可以正常显示顾客信息列表页面。

7.2.5 安全性测试

安全性测试是测试系统在应付非授权的内/外部访问、非法侵入或故意的损坏时的系统防护能力，检验系统是否有能力使可能存在的内/外部的伤害或损害的风险限制在可接受的水平内。可靠性通常包括安全性，但是软件的可靠性不能完全取代软件的安全性，安全性还涉及到数据加密、保密、存取权限等多个方面。

进行安全性测试时需要设计一些测试用例试图突破系统的安全保密措施，检验系统是

否有安全保密漏洞，验证系统的保护机制是否能够在实际使用中不受到非法侵入。在安全测试过程中，测试者扮演成试图攻击系统的角色，尝试获取系统密码，利用能够瓦解任何防守的客户软件攻击系统；或者把系统“制服”，使别人无法访问。

7.2.6 文档测试

为使软件文档能起到多种桥梁的作用，使它有助于程序员编制程序，有助于管理人员监督和管理软件的开发，有助于用户了解软件的工作和应做的操作，有助于维护人员进行有效的修改和扩充，文档的编制必须保证一定的质量。

(1) 针对性：文档编制以前应分清读者对象。按不同的类型、不同层次的读者，决定怎样适应他们的需要。例如，管理文档主要是面向管理人员的，用户文档主要是面向用户的，这两类文档不应像开发文档(面向开发人员)那样过多使用软件的专用术语。

(2) 精确性：文档的行文应当十分确切，不能出现多义性的描述。同一课题几个文档的内容应当是协调一致、没有矛盾的。

(3) 清晰性：文档编写应力求简明，如有可能，配以适当的图表，以增强其清晰性。

(4) 完整性：任何一个文档都应当是完整的、独立的、自成体系的。例如，前言部分应作一般性介绍，正文给出中心内容，必要时还有附录，列出参考资料等。

同一课题的几个文档之间可能有些部分内容相同，这种重复是必要的。不要在文档中出现转引其它文档内容的情况。例如，一些段落没有具体描述，而用“见××文档××节”的方式，这将给读者带来许多的不便。

(5) 灵活性：各个不同软件项目，其规模和复杂程度有着许多实际差别。实际工作中可参阅以下原则：

① 根据具体的软件开发项目，决定编制的文档种类。

软件开发的管理部门应该根据本部门承担的应用软件的专业领域，制定一个文档编制的实施规定。

对于一个具体的应用软件项目，项目负责人应根据上述实施规定，确定一个文档编制计划。其中包括：应该编制的文档和详细程度；各个文档的编制负责人和进度要求；审查、批准的负责人和时间进度安排；在开发时期内各文档的维护、修改和管理的负责人，以及批准手续。

② 开发的软件系统非常大时，一种文档可以分成几卷编写。

项目开发计划可分为质量保证计划、配置管理计划、用户培训计划、安装实施计划等。系统设计说明书可分为系统设计说明书、子系统设计说明书。程序设计说明书可分为程序设计说明书、接口设计说明书、版本说明。操作手册可分为操作手册、安装实施过程。测试计划可分为测试计划、测试设计说明、测试规程、测试用例。测试分析报告可分为综合测试报告、验收测试报告。项目开发总结报告也可分成项目开发总结报告、资源环境统计。

③ 根据任务的规模、复杂性、项目负责人对该软件的开发过程及运行环境所需详细程度的判断，确定文档的详细程度。

④ 文档内容的繁简可根据国标 GB 8567—88《计算机软件产品开发文件编制指南》指导进行，其中关于所建议的所有条款都可以扩展，进一步细分；反之，如果条款中有些细

节并非必需，也可以根据实际情况进行压缩合并。

⑤ 程序的设计表现形式可以使用程序流程图、判定表、程序描述语言(PDI)和问题分析图(PAD)等。

⑥ 对于文档的表现形式，没有规定或限制，可以使用自然语言，也可以使用形式化的语言。

⑦ 当国标 GB 8567—88 中所规定的文档种类不能满足某些应用部门的特殊需要时，可以建立一些特殊的文档种类要求。这些要求可以包含在本单位的文档编制实施规定中。

(6) 可追溯性：由于各开发阶段编制的文档与各个阶段完成的工作有密切的关系，因此文档具有一定的继承关系，项目各个开发阶段之间提供的文档必定存在着可追溯的关系。例如，系统的软件需求必定在设计说明书、测试计划甚至用户手册中有所体现。

7.3 性能测试的步骤

针对不同的系统架构，开发人员可能选择不同的实现方式。下面介绍一种关于如何选择测试策略的方法，帮助分析软件系统的整体架构的性能指标和性能瓶颈，其步骤如下所示：

步骤 1：制定目标和分析系统；

步骤 2：选择测试度量方法；

步骤 3：学习相关技术和工具；

步骤 4：制定评估标准；

步骤 5：设计测试用例；

步骤 6：运行测试用例；

步骤 7：分析测试结果。

1. 制定目标和分析系统

性能测试计划中第一步都会制定目标和分析系统。只有明确目标和了解系统构成才会澄清测试范围，知道在测试中要掌握什么样的技术。明确目标是指确定客户需求和期望、实际业务需求和系统需求。

系统组成包含系统类别、系统构成和系统功能等。

1) 系统类别

分清系统类别是掌握什么样的技术的前提。例如：系统类别是 B/S 结构，需要掌握 HTTP 协议、Java、HTML 等技术；若系统为 C/S 结构，需要了解 OS、Winsock、COM 等。

2) 系统构成

不同的系统构成，性能测试就会得到不同的结果。硬件设置、操作系统设置是性能测试的制约条件，一般性能测试都是利用测试工具模仿大量的实际用户操作，系统在超负荷情形下运作。

3) 系统功能

系统功能是性能测试中要模拟的环节，是指系统提供的不同子系统，如办公管理系统中的公文子系统、会议子系统等。

2. 选择测试度量方法

经过第一步，将会对系统有清醒的认识。接下来进行软件度量，收集系统相关的数据。度量包括如下内容：

- 制定规范
- 制定相关流程、角色、职责
- 制定改进策略
- 制定结果对比标准

3. 学习相关技术和工具

性能测试是通过工具模拟大量用户操作，对系统增加负载，所以必须熟练地掌握和运用测试工具。性能测试工具一般基于不同的软件系统架构实现，其脚本语言也不同。

由于每一种性能测试工具都有自身的特点，因此只有经过工具评估，才能选择符合现有软件架构的性能测试工具，确定测试工具后，需要组织测试人员学习工具的使用，培训相关的测试技术。

4. 制定评估标准

任何测试的目的都是确保软件符合预先规定的目标和要求。通常性能测试有如下 4 种模型技术用于评估。

(1) 线性投射。通过大量的、过去的、扩展的或者将来可能发生的数据组成散布图，利用这个图表不断和系统的当前状况进行对比。

(2) 分析模型。通过预测响应时间，将工作量的数据和系统本质关联起来，进行模型分析。

(3) 模仿。模仿实际用户的使用方法，反复地测试系统。

(4) 基准。定义测试作为标准，与后面进行的测试结果进行对比。

5. 设计测试用例

设计测试用例的原则是以最小的代价提供最多的测试信息，设计测试用例的目标是一次尽可能地包含多个测试要素，这些测试用例必须是测试工具可以实现的，不同的测试场景将测试不同的功能。

6. 运行测试用例

通过性能测试工具运行测试用例，需要不同的测试环境以及不同的机器配置。

7. 分析测试结果

运行测试用例后，收集相关信息，进行数据统计分析，找到性能瓶颈。通过排除误差和其它因素，让测试结果体现真实情况。不同的体系结构所采用的测试方法也不同，B/S 结构的系统通常会分析网络带宽和流量对用户操作响应的影响，而 C/S 结构可能更关心系统整体配置对用户操作的影响。

7.4 网站测试

作为基于 Web 的软件系统，网站测试与传统的软件测试不同：不但要检查和验证网站

是否按照设计的要求运行，还要测试网站是否适合不同用户的浏览器显示，并要从最终用户的角度进行安全性和可用性的各项测试。

7.4.1 网站体系结构

网站属于客户/服务器软件类别，相对于一般的窗口软件有其明显的特点，如图 7.4 所示，网站的结构模型由以下几部分组成。

(1) 第一部分是用户界面，提供使用者交互操作平台，用于数据输入和信息获取。

(2) 第二部分负责将输入数据进行处理，送到下一层，也负责将下一层传回的数据显示在用户界面。

(3) 第三部分负责将上一层传来的数据按照需求规格中特定的业务规则及设定的逻辑进行处理，传送到数据存储层。

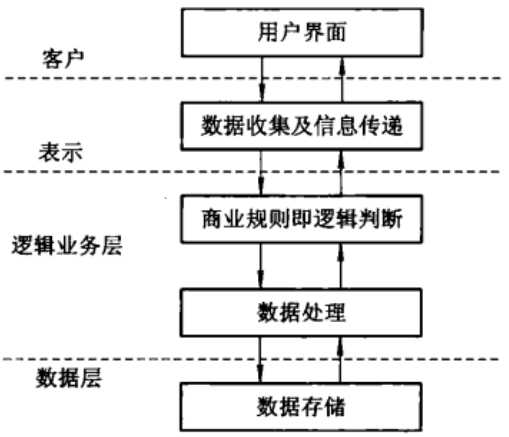


图 7.4 网站的结构模型

(4) 第四部分是数据存储，一般使用数据库方式。

7.4.2 网站测试内容

在网站测试中主要进行以下项目的测试：

(1) UI 测试。UI 测试主要测试页面是否美观(包括页面的布局是否合理，策划是否舒服美观，页面长度是否合理，前景色与背景色是否搭配，页面风格是否统一)。

(2) 链接测试。链接测试用于测试点击链接时是否可以进入所找的页面，是否能正确返回，链接页面会不会是空白页面、孤立页面或根本没链接(也就是说链接的是自己本身)。如果链接的是空白页，我们是否可以正确返回；如果使用了框架或内嵌框架，是否可以正确在本框架页内显示要查找的页面；使用内容置顶时是否可以正确实现。

(3) 表单测试。表单测试包括单选按钮、复选框、文本框、密码项、菜单项和提交按钮类按钮的测试以及后台数据库的测试。

(4) 兼容性测试。兼容性测试是指在各种配置不同的操作系统上和分辨率不同的电脑上及使用不同的浏览器对其测试，看其是否可以正确显示，是否有图片和页面错位或太大太小等问题使有的部分无法看到，是否有图片或视频无法显示等。

(5) 网络配置测试。网络配置测试主要测试网页是否可以打印或保存(如果是保密的网页或不想让别人保存的页面可以将其做成 Flash 格式的，不让用户保存)，网页冗余代码是否过多或容量太大导致网络运行速度过慢等。

(6) 负载测试。负载测试主要测试多个用户同时上网时其最大的承受能力是多大，如果超过了这个极限会有何反应。

(7) 安全测试。安全测试主要测试用户名和密码是否有长度限制，是否有复杂度限制，登录次数是否受限等。

(8) 接口测试。

7.5 思考与习题

一、判断题

1. 负载测试是验证要检验的系统的能力最高能达到什么程度。(　　)
2. 所有软件必须进行某种程度的兼容性测试。(　　)
3. 以消除瓶颈为目的的测试是覆盖测试。(　　)

二、简答题

1. 请解释如下名词概念。

响应时间　并发用户数　吞吐量　性能计数器　休眠时间　点击率

2. 负载测试与压力测试有哪些异同点？请举例说明。
3. 什么是可靠性测试？请举例说明。
4. 什么是安全性测试？它与可靠性测试有什么区别？
5. 文档的编制有什么特点？
6. 网站测试内容有哪些？

第 8 章　面向对象测试

本章主要介绍面向对象测试的基本方法和模型。就面向对象分析测试、面向对象设计测试和面向对象编程测试分别给出了详细说明，并对面向对象的单元测试、集成测试和系统测试给出了解释。

8.1　面向对象影响测试

传统的测试软件是从“小型测试”开始，逐步过渡到“大型测试”，即从单元测试开始，逐步进入集成测试，最后进行确认测试和系统测试。对于传统的软件系统来说，单元测试集中测试最小的可编译的构件单元(模块)；单元测试结束之后，集成到系统中进行一系列的回归测试，以便发现模块接口错误和新单元加入到系统中来所带的副作用；最后，把系统作为一个整体来测试，以发现软件需求中的错误。

面向对象的软件结构与传统的功能模块结构有所区别，类作为构成面向对象程序的基本元素，封装了数据及作用在数据上的操作。父类定义共享的公共特征，子类除继承父类所有特征外，还引入了新的特征。

面向对象技术具有信息隐蔽、封装、继承、多态和动态绑定等特性，提高了软件开发质量，但同时也给软件测试提出了新的问题，增加了测试的难度。

传统软件的测试往往关注模块的算法细节和模块接口间流动的数据，面向对象软件的类测试由封装在类中的操作和类的状态行为所驱动。下面具体分析面向对象技术对软件测试的影响。

8.1.1　封装性影响测试

类的重要特征之一是信息隐蔽，它通过对象的封装性实现。封装将一个对象的各个部分聚集在一个逻辑单元内，对象的访问被限制在一个严格定义的接口上，信息隐蔽只让用户知道某些信息，其它信息被隐藏起来。信息隐蔽与封装性限制了对象属性对外界的可见性与外界对它的操作权限，使得类的具体实现与它的接口相分离，降低类和程序其它各部分之间的依赖，促进程序的模块化，避免外界对其不合理操作并防止错误的扩散。

信息隐蔽给测试带来许多问题。在面向对象软件中，对象行为是被动的，在接受到相关外部信息后才被激活，进行相关操作返回结果。在工作过程中，对象的状态可能发生变化而进入新的状态。通过发送一系列信息创建和激活对象，看其是否完成预期操作并处于正确状态，但是由于信息隐蔽与封装机制，类的内部属性和状态对外界是不可见的，只能通过类自身的方法获得，这给类测试时测试用例执行是否处于预期状态的判断带来困难，

在测试时添加一些对象的实现方式和内部状态的函数考察对象的状态变化。

8.1.2 继承性影响测试

在面向对象程序中，继承由扩展、覆盖和特例化三种基本机制实现。其中扩展是子类自动包含父类的特征；覆盖是子类的方法与父类的方法有相同的名字和消息参数，但其实现的方法不同；特例化是子类中特有的方法和实例变量。继承有利于代码的复用，但同时也使错误传播概率提高。继承使得测试遇见如此难题：对于未重定义的继承特征是否进行测试？对于子类中新添加和重定义的特征如何进行测试?”等等。

Weyuker 提出了基于程序测试数据集的充分性公理，如下所示。

1) 反扩展性公理

反扩展性公理认为若有两个功能相同而实现不同的程序，对其中一个程序是充分的测试数据集未必对另一个也是充分的测试数据集。这一公理表明若在子类中重定义了某一继承的方法，即使两个函数完成相同的功能，对被继承方法是充分的测试数据集未必对重定义的方法是充分的。

2) 反分解性公理

反分解性公理认为一个程序进行过充分的测试，并不表示其中的成分都得到了充分的测试。因为这些独立的成分有可能被用在其它的环境中，此时就需要在新的环境中对这个部分重新进行测试。因此，若一个类得到了充分的测试，当其被子类继承后，继承的方法在子类的环境中的行为特征需要重新测试。

3) 反组合性公理

反组合性公理认为一个测试数据集对于程序中各个单元都是充分的并不表示它对整个程序是充分的，因为独立部分交互时会产生在隔离状态下所不具备的新特性。这一公理表明，若对父类中某一方法进行了重定义，仅对该方法自身或其所在的类进行重新测试是不够的，还必须测试其它有关的类(如子类和引用类)。

Perry 和 Kaiser 对 Weyuker 观点总结如下：有关充分测试的直觉的结论可能是错误的。随着继承层次的加深，虽然可供重用的类越来越多，编程效率也越来越高，但无形中加大了测试的工作量和难度。同时，递增式软件开发过程中，如果父类发生修改，这种变化会自动传播到所有子类，使得父类和子类都必须重新测试。所以说，继承并未简化测试问题，反而使测试更加复杂。

8.1.3 多态性影响测试

多态使得面向对象程序对外呈现出强大的处理能力，但同时却使得程序内“同一”函数的行为复杂化，多态促成了子类型替换。一方面，子类型替换使对象的状态难以确定。如果一个对象包含了 A 类型的对象变量，则 A 类型的所有子类型的对象也允许赋给该变量。程序运行过程中，该变量可能引用不同类型的对象，其结构不断变化。另一方面，子类型替换使得向父类对象发送的消息也允许向该类的子类对象发送。如果 A 类有两个子类 B 和 C，D 类也有两个子类 E 和 F，A 类对象向 D 类对象发送消息 m，则测试 A 类对象发出的消息 m 时，需考虑所有可能的组合。

由此可见，多态性和动态绑定使得系统能自动为给定消息选择合适的实现代码，但它所带来的不确定性，使得传统测试遇到障碍，增加了测试用例的选取难度。

8.2　面向对象测试模型

面向对象开发分为面向对象分析、面向对象设计和面向对象编程三个阶段。分析阶段产生整个问题空间的抽象描述，设计出类和类结构，最后形成代码。面向对象测试模型能有效地将分析、设计的文本或图表代码化。测试模型如图 8.1 所示。

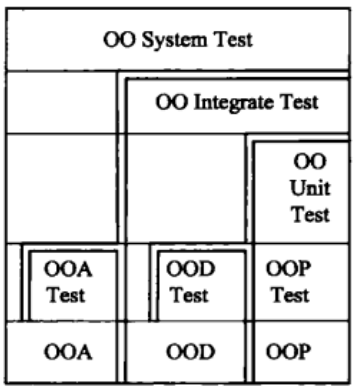


图 8.1　面向对象测试模型

其中，OOA Test 是面向对象分析测试；OOD Test 是面向对象设计测试，OOA Test 和 OOD Test 主要面向的是分析设计文档，是软件开发前期的关键性测试。

OOP Test 是面向对象编程测试，主要针对编程风格和程序代码进行测试。OO Unit Test 指面向对象单元测试，对程序内部单一的功能模块测试，是面向对象集成测试的基础；OO Integrate Test 是指面向对象集成测试，主要对系统内部的相互服务进行测试，如成员函数间的相互作用、类之间的消息传递等；OO System Test 是指面向对象系统测试，主要以需求规格说明为测试标准。

8.3　面向对象分析测试

传统面向过程分析是功能分解的过程，着眼点在于一个系统需要什么样的信息处理方法和过程。而 OOA 直接映射需求分析问题，将问题空间功能抽象化，用对象的结构反映实例和实例之间的复杂关系，OOA 为类的实现以及类层次结构的组织和实现提供平台。

OOA Test 分为五个方面：对象测试、结构测试、主题测试、属性和实例关联测试、服务和消息关联测试。

1. 对象测试

OOA 中认定的对象是对问题空间中实例的抽象，可从以下方面对其进行测试：

(1) 认定的对象是否全面，问题空间中所有涉及到的实例是否都反映在认定的抽象对象中。

(2) 认定的对象是否具有多个属性。只有一个属性的对象通常应看成其它对象的属性，

而不是抽象为独立的对象。

(3) 对认定为同一对象的实例是否有共同的、区别于其它实例的共同属性。

(4) 对认定为同一对象的实例是否提供或需要相同的服务，如果服务随着不同的实例而变化，认定的对象就需要分解或利用继承性来分类表示。

(5) 系统没有必要始终保持对象代表的实例信息，提供或者得到关于它的服务，认定的对象也无必要。

(6) 认定的对象的名称应该尽量准确、适用。

2. 结构测试

结构作为对象的组织方式，用来反映问题空间中的复杂实例和复杂关系。认定的结构分为两种：分类结构和组装结构。其中，分类结构体现了问题空间中实例的一般与特殊关系，组装结构体现了问题空间中实例整体与局部的关系。

1) 对认定的分类结构的测试

(1) 对于结构中处于高层的对象，是否在问题空间中含有不同于下一层对象的特殊可能性，即是否能派生出下一层对象。

(2) 对于结构中处于同低层的对象，是否能抽象出在现实中有意义的更一般的上层对象。

(3) 对所有认定的对象，是否能在问题空间内抽象出在现实中有意义的对象。

(4) 高层的对象的特性是否完全体现下层的共性。

(5) 低层的对象是否具有高层对象特性基础上的特殊性。

2) 对认定的组装结构的测试

(1) 整体和部件的组装关系是否符合现实的关系。

(2) 整体和部件是否在考虑的问题空间中有实际应用。

(3) 整体中是否遗漏了反映在问题空间中有用的部件。

(4) 部件是否能够在问题空间中组装新的有现实意义的整体。

3. 主题测试

主题如同文章中的内容概要，是在对象和结构基础上抽象，其提供 OOA 分析结果的可见性。主题测试应该考虑以下方面：

(1) 贯彻 George Miller 的“7+2”原则，如果主题个数超过 7 个，就要对相关密切的主题进行归并。

(2) 主题所反映的一组对象和结构是否具有相同和相近的属性及服务。

(3) 认定的主题是否是对象和结构更高层的抽象，是否便于理解 OOA。

(4) 主题间的消息联系是否代表了主题所反映的对象和结构之间的所有关联。

4. 属性和实例关联测试

属性用来描述对象或结构所反映的实例特性。实例关联用于反映实例集合间的映射关系。对属性和实例关联的测试从如下方面考虑：

(1) 定义的属性是否对相应的对象和分类结构的每个实例都适用。

(2) 定义的属性在现实世界是否与这种实例关系密切。

(3) 定义的属性在问题空间是否与这种实例关系密切。

(4) 定义的属性是否能够不依赖于其它属性被独立理解。

(5) 定义的属性在分类结构中的位置是否恰当，低层对象的共有属性是否在上层对象属性体现。

(6) 在问题空间中每个对象的属性是否定义完整。

(7) 定义的实例关联是否符合现实。

(8) 在问题空间中实例关联是否定义完整，特别需要注意一对多和多对多的实例关联。

5. 服务和消息关联测试

服务定义了每种对象和结构在问题空间所要求的行为。问题空间中实例的通信在 OOA 中相应地定义为消息关联。对定义的服务和消息关联的测试可从如下方面进行：

(1) 对象和结构在问题空间的不同状态是否定义了相应的服务。

(2) 对象或结构所需要的服务是否都定义了相应的消息关联。

(3) 定义的消息关联所指引的服务提供是否正确。

(4) 沿着消息关联执行的线程是否合理，是否符合现实过程。

(5) 定义的服务是否重复，是否定义了能够得到的服务。

面向对象分析测试如表 8.1 所示。

表 8.1　面向对象分析测试

测试内容	概　述		测试考虑方面
认定对象的测试	认定的对象：对问题空间中的结构、其它系统、设备、被记忆的事件、系统涉及的人员等进行实际的抽象		(1) 是否全面，问题空间中的实例是否都反映在认定的抽象对象中；(2) 是否具有多个属性，只具有一个属性的对象不抽象为独立的对象；(3) 对认定为同一对象的实例是否有区别于其它实例的共同属性；(4) 对认定为同一对象的实例是否提供或需要相同的服务，如果服务随着实例的不同而变化，认定的对象就需要分析或利用继承性来分类表示；(5) 如果系统没有必要始终保持对象代表的实例信息，提供或者得到关于它的服务、认定的对象也无必要；(6) 认定的对象的名称要尽量准确、适用
认定结构的测试	认定结构：多种对象的组织方式，用来反映问题空间中的复杂实例和复杂关系，认定的结构分为两种：分类结构和组装结构	分类结构：体现问题空间中实例的一般与特殊关系	(1) 结构中的一种对象尤其是高层对象，是否存在不同于下一层对象的特殊的可能性，即是否能派生出下一层对象；(2) 结构中的一种对象尤其是同一低层对象，是否能抽象出在现实中有意义的更一般的上层对象；(3) 对所有认定的对象，是否能向上层抽象出现实中有意义的对象；(4) 高层的对象的特性是否完全体现下层的共性；(5) 低层的对象是否有高层特性基础上的特殊性
		组装结构：体现问题空间中实例的整体与局部的关系	(1) 整体对象和部件对象的组装关系是否符合现实的关系；(2) 整体对象和部件对象是否在考虑的问题空间中有实际关系；(3) 整体对象是否遗漏了在问题空间中有用的部件对象；(4) 部件对象是否能够在问题空间中组装成新的有意义的整体对象

续表

测试内容	概　述	测试考虑方面
认定主题的测试	主题：在对象和结构的基础上更高一层的抽象，是为了提供OOA分析结果的可见性，如同文章中各章的摘要	(1) 贯彻George Mille的“7+2”原则。如果主题个数A超过7个，则对有较密切属性和服务的主题归并；(2) 主题所反映的一组对象和结构是否具有相同或相近的属性和服务；(3) 认定的主题是否是对象和结构更高一层的抽象，是否便于理解OOA结果的概括；(4) 主题间的消息联系(抽象)是否代表主题所反映的对象和结构之间的所有关联
对定义的属性和实例关联的测试	属性：用来描述对象或结构所反映的实例的特性； 实例关联：反映实例集合间的映射关系	(1) 定义的属性是否对相应的对象和分类结构的每个现实实例都适用；(2) 定义的属性在现实世界是否与这种实例关系密切；(3) 定义的属性在问题空间是否与这种实例关系密切；(4) 定义的属性是否能够不依赖于其它属性被独立理解；(5) 定义的属性在分类结构中的位置是否恰当，低层对象的共有属性是否在上层对象属性中体现；(6) 在问题空间中每个对象的属性是否定义完整；(7) 定义的实例关联是否符合现实；(8) 在问题空间中实例关联是否定义完整，特别需要注意“一对多”、“多对多”的实例关联
对定义的服务和消息关联的测试	定义的服务：定义的每一种对象和结构在问题空间所要求的行为； 消息关联：问题空间中实例之间必要的通信，需要定义相应的消息关联	(1) 对象和结构在问题空间中的不同状态是否定义相应的服务；(2) 对象和结构所需的服务是否都定义了相应的消息关联；(3) 定义的消息关联所指引的服务是否正确；(4) 沿着消息关联执行的线程是否合理，是否符合现实过程；(5) 定义的服务是否重复，是否定义了能够得到的服务

8.4 面向对象设计测试

结构化设计方法采用面向作业的设计方法，把系统分解为一组作业。面向对象设计采用“造型的观点”，是以OOA为基础归纳出类，建立类结构，实现分析结果对问题空间的抽象，设计类的服务。由此可见，OOD是OOA的进一步细化和抽象，其界限通常难以严格区分。OOD确定类和类结构不仅是满足当前需求分析的要求，更重要的是通过重新组合或加以适当的补充，实现功能的重用和扩增。

因此，OOD测试可从如下三方面考虑：

(1) 对认定类测试；

(2) 对构造的类层次结构测试；

(3) 对类库的支持测试。

1. 对认定类测试

OOD认定的类是OOA中认定的对象，是对象服务和属性的抽象。认定类应该尽量是基础类，这样便于维护和重用。

测试认定类有如下一些准则：

(1) 是否涵盖了 OOA 中所有认定的对象。
(2) 是否能体现 OOA 中定义的属性。
(3) 是否能实现 OOA 中定义的服务。
(4) 是否对应着一个含义明确的数据抽象。
(5) 是否尽可能少地依赖其它类。

2. 对类层次结构测试

OOD 的类层次结构基于 OOA 的分类结构产生，体现了父类和子类之间的一般性和特殊性。类层次结构是在解空间构造实现全部功能的结构框架。测试包含如下方面：

(1) 类层次结构是否涵盖了所有定义的类。
(2) 是否能体现 OOA 中所定义的实例关联。
(3) 是否能实现 OOA 中所定义的消息关联。
(4) 子类是否具有父类没有的新特性。
(5) 子类间的共同特性是否完全在父类中得以体现。

3. 对类库支持测试

类库主要用于支持软件开发的重用，对类库的支持属于类层次结构的组织问题。由于类库并不直接影响软件的开发和功能实现，因此类库的测试往往作为对高质量类层次结构的评估。其测试点如下：

(1) 一组子类中关于某种含义相同或基本相同的操作是否有相同的接口。
(2) 类中方法功能是否较单纯，相应的代码行是否较少，一般建议不超过 30 行。
(3) 类的层次结构是否深度大、宽度小。

8.5　面向对象单元测试

面向对象软件测试过程以层次增量的方式进行。首先，对类方法进行测试；然后，对类进行测试；再次，将多个类集成为类簇或子系统进行集成测试；最后，进行系统测试。其中，面向对象单元测试针对类中的成员函数以及成员函数间的交互进行测试；面向对象集成测试主要对系统内部的相互服务进行测试，如类之间的消息传递等；面向对象系统测试是基于面向对象集成测试的最后阶段的测试，主要以用户需求为测试标准。

下面介绍类的测试方法。

8.5.1　功能性和结构性测试

类测试有两种主要的方式：功能性测试和结构性测试。功能性测试和结构性测试分别对应传统测试的黑盒测试和白盒测试。功能性测试以类的规格说明为基础，主要检查类是否符合规格说明的要求，包括类的规格说明和方法的规格说明两个层次。例如，对于 Stack 类，检查操作是否满足 LIFO 规则。结构性测试从程序出发，对方法进行测试，考虑代码是否正确，Stack 类检查代码是否执行正确且至少执行过一次。

测试类的方法指对方法调用关系进行测试。测试每个方法的所有输入情况，并对这些方法之间的接口进行测试。对类的构造函数参数以及消息序列进行选择以保证其在状态集

合下正常工作。因此，对类的测试分成如下两个层次：方法内测试和方法间测试。

1) 方法内测试

方法内测试考虑类中方法，等效于传统程序中单个过程的测试，传统测试技术(如逻辑覆盖、等价类划分、边界值分析和错误推测等方法)仍然作为测试类中每个方法的主要手段。与传统单元测试的最大差别在于方法内测试改变了它所在实例的状态，这就要求对隐藏的状态信息进行评估。

面向对象软件中方法的执行是通过消息驱动执行的。测试类中的方法，必须用驱动程序对被测方法通过发送消息来驱动执行。如果被测试模块或者方法调用其它模块或方法，则需要设计一个模拟被调程序功能的存根程序代替。驱动程序、存根程序及被测模块或方法组成一个独立的可执行单元。

2) 方法间测试

方法间测试考虑类中方法之间的相互作用，对方法进行综合测试。单独测试一个方法时，只考虑其本身执行的情况，而没有考虑方法的协作关系。方法间测试考虑一个方法调用本对象类中的其它方法，或其它类的方法之间的通信情况。

类的操作被封装在类中，对象之间通过发送消息启动操作，对象作为一个多入口模块，必须考虑测试方法的不同次序组合的情况。当一个类中方法的数目较多时，次序的组合数目将非常多。对于操作的次序组合以及动作的顺序问题，测试用例中加入了激发调用信息，检查它们是否正确运行。对于同一类中方法之间的调用，遍历类的所有主要状态。同时，选出最可能发现属性和操作错误的情况，重点进行测试。

8.5.2 测试用例的设计和选择

1. 测试用例设计

传统软件测试用例设计从软件的各个模块算法出发，而面向对象(OO)软件测试用例着眼于操作序列，以实现对类的说明。

OO 测试用例设计对 OO 的五个特性(局域性、封装性、信息隐藏、继承性和抽象)进行测试。Berard 提出了测试用例的设计方法，关于设计合适的操作序列以测试类的状态，主要原则包括:

(1) 对每个测试用例应当给予特殊的标识，并且还应当与测试的类有明确的联系。

(2) 测试目的应当明确。

(3) 应当为每个测试用例开发一个测试步骤列表，列表包含以下内容：

① 列出所要测试对象的说明；

② 列出将要作为测试结果的消息和操作；

③ 列出测试对象可能发生的例外情况；

④ 列出外部条件，为了正确对软件进行测试所必需的外部环境的变化；

⑤ 列出为帮助理解和实现测试所需要的附加信息。

2. 基于概率分布的测试用例抽样

总体是指所有可能被执行的测试用例，包括所有前置条件和所有输入值可能的组合情况。样本是基于概率分布选择的子集，子集的使用频率越高，被选中的概率越大。

样本集合中每个样本代表一个特定的个体。例如，用例模型作为测试用例分层的基础，挑选出一个测试用例的抽样，选择一个测试系列，并不要求一定要首先明确如何来确定测试用例的总体。构建测试用例的一个测试系列，将类说明作为测试用例的来源，运用一种抽样方法对测试进行补充，减少测试的数目。

【例 8-1】　类的实例变量取值范围为 0～359，采用相关测试方法设计测试用例。

【解答】　(1) 采用基于边界值的测试方法，取 0 值周围的 −1、0、1 三种测试和 359 附近的 358、359、400 等测试用例。(2) 采用随机测试方法，使用随机数发生器 random()。由于每个测试用例的抽样为(0～359)，则设计 int(random()*360)和 int(−1*random()*360)进行随机的抽样，每个值都在该区间内，且每个值被选中的概率相等，测试不同的值。

8.6　面向对象集成测试

8.6.1　概述

传统面向过程的软件模块具有层次性，模块之间存在着控制关系。面向对象软件功能散布在不同类中，通过消息传递提供服务。由于面向对象软件没有一个层次的控制结构，传统软件自顶向下和自底向上的组装策略意义不大，构成类的各个部件之间存在直接和非直接交互，软件的控制流无法确定，采用传统的将操作组装到类中的增值式组装常常行不通。

集成测试关注于系统的结构和类之间的相互作用，测试步骤一般分成两步，首先进行静态测试，然后进行动态测试。静态测试主要针对程序的结构进行，检测程序结构是否符合设计要求，采用逆向工程测试工具得到类的关系图和函数关系图，与面向对象设计规格说明比较检测程序结构和实现上是否有缺陷，是否符合需求设计。

动态测试根据功能结构图、类关系图或者实体关系图，确定不需要被重复测试的部分，通过覆盖标准减少测试工作量。覆盖标准有如下几类：

(1) 达到类所有的服务要求或服务提供的覆盖率。

(2) 依据类之间传递的消息，达到对所有执行线程的覆盖率。

(3) 达到类的所有状态的覆盖率。

通过下列步骤设计测试用例：

(1) 选定检测的类，参考 OOD 分析结果，得到类的状态和行为、类或成员函数间传递的消息、输入或输出的界定等数据。

(2) 确定采用什么样的覆盖标准。

(3) 利用结构关系图确定待测类的所有关联。

(4) 根据程序中类的对象构造测试用例，确认使用什么输入激发类的状态、使用类的服务和期望产生什么行为等。

8.6.2　面向对象交互测试

1．概述

面向对象软件由若干对象组成，通过对象之间的相互协作实现既定功能。交互既包含

对象和其组成对象之间的消息，还包含对象和与之相关的其它对象之间的消息，是一系列参与交互的对象协作中的消息的集合。例如，对象作为参数传递给另一对象，或者当一个对象包含另一对象的引用并将其作为这个对象状态的一部分时，对象的交互就会发生。

对象交互的方式有如下几类：

(1) 公共操作将一个或多个类命名为正式参数的类型。

(2) 公共操作将一个或多个类命名为返回值的类型。

(3) 类的方法创建另一个类的实例，并通过该实例的调用操作。

(4) 类的方法引用某个类的全局实例。

交互测试的重点是确保对象之间进行消息传递，当接收对象的请求，处理方法的调用时，由于可能发生多重的对象交互，因此需要考虑交互对象内部状态的影响，以及相关对象的影响。这些影响主要包括：所涉及的对象的部分属性值的变化，所涉及的对象的状态的变化，创建一个新对象和删除一个已经存在的对象而发生的变化。

交互测试具有以下几个特点：

(1) 假定相互关联的类都已经被充分测试。

(2) 交互测试建立在公共操作上，相对于建立在类实现的基础上要简单。

(3) 采用一种公共接口方法，将交互测试限制在与之相关联的对象上。

(4) 根据每个操作说明选择测试用例，并且这些操作说明都基于类的公共接口。

2. 交互类型

面向对象程序中的类分为原始类和非原始类。原始类是最简单的组件，其数目较少。非原始类是指在某些操作中支持或需要使用其它对象的类。根据非原始类与其它实例交互的程度，非原始类分为汇集类和协作类。下面具体介绍汇集类和协作类测试。

1) 测试汇集类

汇集类是指有些类的说明中使用对象，但是实际上从不和这些对象进行协作。编译器和开发环境的类库通常包含汇集类。例如，C++的模板库、列表、堆栈、队列和映射等管理对象。汇集类一般具有如下行为：

(1) 存放这些对象的引用；

(2) 创建这些对象的实例；

(3) 删除这些对象的实例。

2) 测试协作类

凡不是汇集类的非原始类就是协作类。协作类是指在一个或多个操作中使用其它的对象并将其作为实现中不可缺少的一部分。协作类测试的复杂性远远高于汇集类的测试，协作类测试必须在参与交互的类的环境中进行测试，需要创建对象之间交互的环境。

3. 交互测试

系统交互既发生在类内方法之间，也发生在多个类之间。类 A 与类 B 交互如下所述：

(1) 类 B 的实例变量作为参数传给类 A 的某方法，类 B 的改变必然导致对类 A 的方法的回归测试。

(2) 类 A 的实例作为类 B 的一部分，类 B 对类 A 中变量的引用需进行回归测试。

交互测试的粒度与缺陷的定位密切相关，粒度越小越容易定位缺陷。但是，粒度小使

得测试用例数和测试执行开销增加。因此，测试权衡于资源制约和测试粒度之间，应正确地选择交互测试的粒度。

被测交互聚合块大小的选择，需要考虑以下三个因素：

(1) 区分那些与被测对象有组成关系的对象和那些仅仅与被测对象有关联的对象。在类测试期间，测试组合对象与其组成属性之间的交互。集成测试时，测试对象之间的交互。

(2) 交互测试期间所创建的聚合层数与缺陷的能见度紧密相关，若“块”太大，会有不正确的中间结果。

(3) 对象关系越复杂，一轮测试之前被集成的对象应该越少。

8.7　面向对象的系统测试

单元测试和集成测试仅能保证软件开发的功能得以实现，不能确认在实际运行时是否满足用户的需要，因此，必须对软件进行规范的系统测试。确认测试和系统测试不关心类之间连接的细节，仅着眼于用户的需求，测试软件在实际投入使用中与系统其它部分配套运行的情况，保证系统各部分在协调工作的环境下能正常工作。

系统测试参照面向对象分析模型，测试组件序列中的对象、属性和服务。组件是由若干类构建的，首先实施接受测试。接受测试将组件放在应用环境中，检查类的说明，采用极值甚至不正确数值进行测试。其次，组件的后续测试应顺着主类的线索进行。

8.8　思考与习题

简答题

1. 什么是汇集类？什么是协作类？怎样测试汇集类和协作类？
2. 类测试的方法有哪些？类测试分为几个层次？
3. 对 OOA 阶段的测试划分为几个方面？分别是什么？
4. 软件测试模型是什么？
5. 测试抽象类有哪些方法？各自的优缺点是什么？

第9章 嵌入式测试

本章介绍了嵌入式软件的测试方法、测试过程、测试特点，嵌入式软件的相关测试工具和测试策略，并讲解了嵌入式软件的测试实例。

9.1 嵌入式软件测试的方法

嵌入式软件测试分为4个阶段，即模块测试、集成测试、系统测试、硬件/软件集成测试。前3个阶段适用于任何软件的测试，硬件/软件集成测试阶段是嵌入式软件所特有的，目的是验证嵌入式软件与其所控制的硬件设备能否正确地交互。

在嵌入式软件测试中，常采取折中的方式。基于目标的测试消耗较多的经费和时间，而基于宿主的测试代价较小，但毕竟是在模拟环境中进行的。目前的趋势是把更多的测试转移到宿主环境中进行，但是，目标环境的复杂性和独特性不可能完全模拟。

在两个环境中可以出现不同的软件缺陷，重要的是目标环境和宿主环境的测试内容有所选择。在宿主环境中，可以进行逻辑或界面的测试以及与硬件无关的测试。在模拟或宿主环境中的测试所消耗的时间通常相对较少，用调试工具可以更快地完成调试和测试任务。而与定时问题有关的白盒测试、中断测试、硬件接口测试只能在目标环境中进行。在软件测试周期中，基于目标的测试是在较晚的硬件/软件集成测试阶段开始的，如果不更早地在模拟环境中进行白盒测试，而是等到硬件/软件集成测试阶段再进行全部的白盒测试，将耗费更多的财力和人力。

9.2 嵌入式软件测试的过程

根据嵌入式系统的开发流程，为了最经济地实现系统的功能，一般采用自顶向下、层层推进的方法对嵌入式系统进行测试。图9.1为基于模块化设计的嵌入式软件测试流程。

嵌入式软件测试的总体步骤为：首先进行操作系统移植并编写系统底层驱动，然后进行系统平台测试，其中包括硬件电路测试、操作系统及底层驱动程序的测试等。如果测试未通过，需要重新进行操作系统移植和编写系统底层驱动；如果此测试通过，可以进入下一步的开发——用模块化的方法编写应用代码，随后再对软件模块进行测试。如果测试没有通过，则要对此代码模块进行修改，然后对软件模块进行测试；如果所有的模块都通过测试，需要进行集成测试。如果集成测试没有通过，则要确定模块接口函数错误模块，然后修改错误模块代码，再利用关联矩阵确定需测试模块，并重新回到软件模块测试；如果集成测试通过，则需要进行系统测试。如果系统测试没有通过，则需要修改程序代码，如果问题出现在操作系统的移植上，需要重新进行操作系统的移植；如果问题只是出现在软

件模块上，只需修改软件模块就行了。如果系统测试通过，就可以退出测试。在第一件产品生产出来之后，需要对产品进行测试，如果测试通过，则表示嵌入式产品的所有测试步骤已经完成。

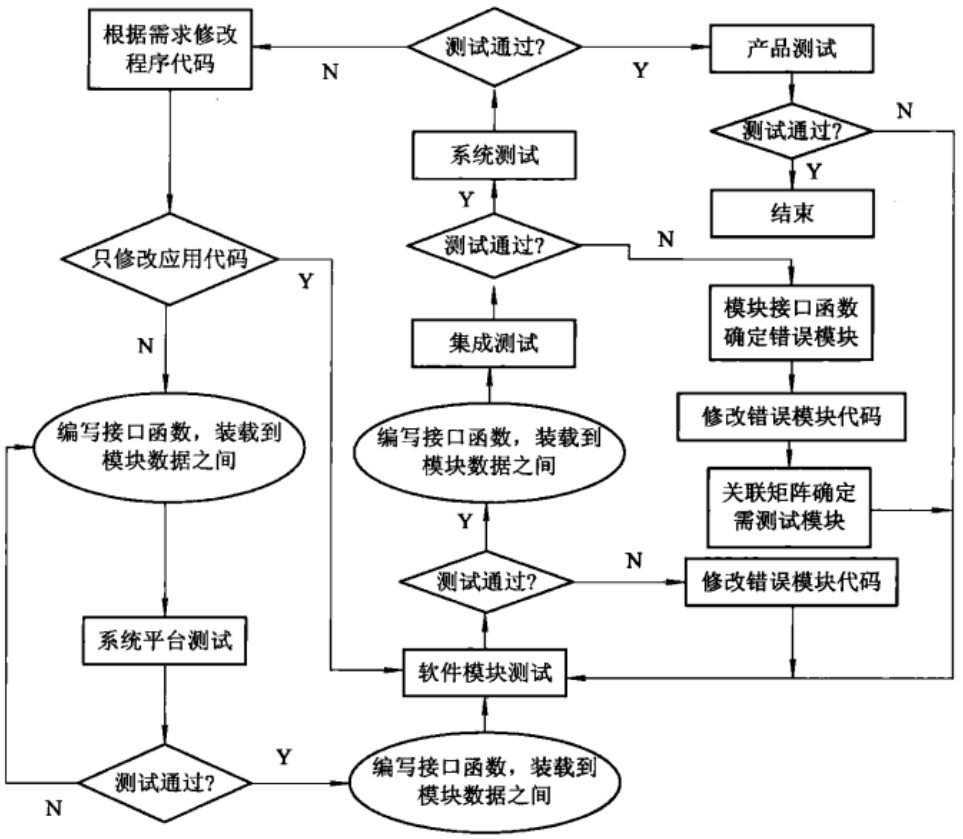


图 9.1 嵌入式软件测试的详细流程

9.3 嵌入式软件测试的特点

嵌入式软件测试作为一种特殊的软件测试，它的目的和原则同普通的软件测试是一样的，都是为了验证或达到可靠性要求而对软件所进行的测试。嵌入式软件测试除了要遵循普通软件测试的原则之外，还需要遵循以下几个原则：

(1) 嵌入式软件测试对软件在硬件平台的测试是必不可少的。

(2) 嵌入式软件测试需要在特定的环境下对软件进行测试。

(3) 嵌入式软件需进行必要的可靠性负载测试，比如测试某些嵌入式系统能否连续 1000 个小时不断电工作。

(4) 除了要对嵌入式软件系统的功能进行测试之外，还需要对实时性进行测试。在判断系统是否失效方面，除了看它的输出结果是否正确，还应考虑其是否在规定的时间里输出了结果。

(5) 在对嵌入式软件进行测试的时候，需要在特定的硬件平台上进行性能测试、内存测试、GUI 测试、覆盖分析测试。

总之，嵌入式软件测试的目的和原则同普通软件测试的目的和原则类似，但在一定程度上要高于普通软件测试的目的和原则。

9.4 嵌入式软件测试的工具

用于辅助嵌入式软件测试的工具很多，下面对几类比较实用的有关嵌入式软件的测试工具加以介绍和分析。

1. 内存分析工具

在嵌入式系统中，内存容量通常是有限的。内存分析工具用来处理在动态内存分配中存在的缺陷。动态内存分配错误，通常是难以复原的，其导致的失效难以追踪，使用内存分析工具可以避免这类缺陷进入功能测试阶段。目前有两类内存分析工具——软件工具和硬件工具。基于软件的内存分析工具可能会对代码的性能造成很大影响，从而严重影响实时操作；基于硬件的内存分析工具价格昂贵，而且只能在工具所限定的运行环境中使用。

2. 性能分析工具

在嵌入式系统中，程序的性能通常是非常重要的。经常会有这样的要求，在特定时间内处理一个中断，或生成具有特定定时要求的一帧。开发人员面临的问题是决定应该对哪一部分代码进行优化来改进性能，并避免花费大量的时间去优化那些对性能没有任何影响的代码。性能分析工具会提供有关的数据，说明执行时间是如何消耗，什么时候消耗的，以及每个例程所用的时间。根据这些数据，确定哪些例程消耗部分执行时间，从而可以决定如何优化软件，以获得更好的时间性能。对于大多数应用来说，大部分执行时间用在相对少量的代码上，费时的代码估计只占所有软件总量的 5%～20%。性能分析工具不仅能指出哪些例程花费了时间，而且与调试工具联合使用可以引导开发人员查看需要优化的特定函数，性能分析工具还可以引导开发人员发现在系统调用中存在的错误以及程序结构上的缺陷。

3. GUI 测试工具

很多嵌入式应用提供了某种形式的图形用户界面，有些系统性能测试是根据用户输入响应时间进行的。GUI 测试工具可以作为脚本工具在开发环境中运行测试用例，其功能包括对操作的记录和回放、抓取屏幕显示供以后分析和比较、设置和管理测试过程。对没有 GUI 的嵌入式设备，可以对其进行插装来运行 GUI 测试脚本，虽然这种方式可能要求对被测代码进行更改，但是节省了功能测试和回归测试的时间。

4. 覆盖分析工具

在进行白盒测试时，可以使用代码覆盖分析工具追踪被执行过的代码。分析过程可以通过插装的方式来完成，插装可以是在测试环境中嵌入硬件，也可以是在可执行代码中加入软件，也可以是二者相结合。测试人员对结果数据加以总结，确定哪些代码被执行过，哪些代码被遗漏了。覆盖分析工具一般会提供有关功能覆盖、分支覆盖、条件覆盖的信息。对于嵌入式软件来说，代码覆盖分析工具可能侵入代码的执行，影响实时代码的运行过程。基于硬件的代码覆盖分析工具的侵入程度要小一些，但是价格一般比较昂贵，而且限制被测代码的数量。

9.5 嵌入式软件测试策略

嵌入式系统的某些自身特点，如实时性(Real-timing)，内存不丰富，I/O 通道少，开发工具昂贵，并且与硬件紧密相关的 CPU 种类繁多等，使得嵌入式软件测试与一般商用软件的开发和测试策略有了很大的不同。

嵌入式软件的开发环境被认为是主机平台，软件运行环境为目标平台，相应的测试为 host-target 测试或 cross-test。

在测试的各个阶段，是采用 host-target 还是 cross-test，应遵循以下策略。

1．单元测试

所有单元测试都可以在主机环境上进行，除非少数情况特别指定了单元测试直接在目标环境进行。测试时，尽可能在主机环境中进行软件测试，通过尽可能小的目标单元访问所有目标指定的界面。

在主机平台上运行的测试速度比在目标平台上快得多，在主机平台完成测试后，可以在目标环境中重复作一简单的确认测试，确认测试结果不受主机和目标机的环境影响。在目标环境中进行确认测试将确定一些未知的、未预料到的、未说明的主机与目标机的不同。例如，目标编译器可能有 bug，但在主机编译器上没有。

2．集成测试

软件集成也可在主机环境上完成，并在主机平台上模拟目标环境运行。当然在目标环境上重复测试也是必需的，在此级别上的确认测试将确定一些环境上的问题，比如内存定位和分配上的一些错误。

在主机环境中如何进行集成测试，取决于目标系统的具体功能有多少。有些嵌入式系统与目标环境耦合得非常紧密，若在主机环境做集成是不切实际的。对于大型软件的开发则可以分为几个级别的集成。低级别的软件集成在主机平台上完成有很大优势，越往后的集成越依赖于目标环境。

3．系统测试和确认测试

所有的系统测试和确认测试都必须在目标环境下执行。当然在主机上开发和执行系统测试，然后移植到目标环境重复执行是很方便的。对目标系统的依赖性会妨碍将主机环境中的系统测试移植到目标系统上，况且只有少数开发者会卷入系统测试，所以有时放弃在主机环境上执行系统测试可能更方便。

确认测试最终的实施舞台必须在目标环境中，系统的确认必须在真实系统之下测试，而不能在主机环境下模拟。这关系到嵌入式软件的最终使用。

使用有效的 cross-test 测试策略可极大地提高嵌入式软件开发测试的水平和效率，当然正确地使用测试工具也是必不可少的。

总之，应用测试工具进行 cross-test 时的策略如下：

(1) 使用测试工具的插装功能(主机环境)执行静态测试分析，并且为动态覆盖测试准备好一插装好的软件代码。

(2) 使用源码在主机环境执行功能测试，修正软件的错误和测试脚本中的错误。

(3) 使用插装后的软件代码执行覆盖率测试，添加测试用例或修正软件的错误，保证达到所要求的覆盖率目标。

(4) 在目标环境下重复(2)，确认软件在目标环境中执行测试的正确性。

(5) 若测试需要达到极端的完整性，最好在目标系统上重复(3)，确定软件的覆盖率没有改变。

通常在主机环境中执行多数的测试，只是在最终确定测试结果和最后的系统测试时才移植到目标环境，这样可以避免发生访问目标系统资源方面的瓶颈，也可以减少使用昂贵资源如在线仿真器等的费用。另外，若目标系统的硬件由于某种原因而不能使用，最后的确认测试可以推迟直到目标硬件可用，这为嵌入式软件的开发测试提供了弹性。设计软件的可移植性是成功进行 cross-test 的先决条件，它通常可以提高软件的质量，并且对软件的维护大有益处。以上所提到的测试工具，都可以通过各自的方式提供测试在主机与目标机之间的移植，从而使嵌入式软件的测试得以方便地执行。

使用有效的 cross-test 测试策略可极大地提高嵌入式软件开发测试的水平和效率，提高嵌入式软件的质量。

9.6 嵌入式软件测试实例

嵌入式软件测试与一般软件测试的最大不同就是必须为嵌入式软件测试搭建一个测试环境。下面以设计一台遥控摄像头的控制软件为例，具体介绍如何建立嵌入式软件的测试环境。

假定这台遥控摄像头不带操作系统，软件的设计是直接基于硬件的。这种情况在建立测试环境时相对比较复杂。在此要进行的是原型阶段的软件集成测试，利用仿真方法通过纯软件仿真的方式建立测试环境。另外，考虑到遥控摄像头的控制及通信软件在实际系统中运行于 Intel 8086 之上，而 8086 的指令系统是目前 PC 所使用指令的子集。因此，结合现有环境和技术，在这次软件测试过程中，我们对嵌入式软件进行编译、插入驱动和插桩模块后移植到 DOS 环境，再配置相应的测试用例，组建成基于 DOS 的测试环境。具体操作可按以下三步进行。

1) 软件指令仿真

软件指令仿真的主要工作是对相关的 I/O 操作进行替换。在 80X86 系列 CPU 指令集中，I/O 指令有两个 IN 和 OUT，对这两个指令，我们都定义相应的宏来代替其操作，同时在内存中组织变量来代替 I/O 操作中的寄存器变量。

在软件中，I/O 指令主要有以下几类：

- IN REGISTER, BYTE
- IN REGISTER, WORD
- OUT BYTE, REGISTER
- OUT REGISTER, WORD

构造如下的宏指令仿真上述指令的功能：

```
int macro reg, port
  mov di, counti
```

```
    mov al, byte ptr inbuf[di]
    inc di
    mov counti, di
    endm

inw macro reg, port
    mov di, counti
    mov ah, byte ptr inbuf[di]
    Inc di
    mov al, byte ptr inibuf[di]
    Inc di
    mov counti, di
    endm

outb macro reg, port
    mov di, counto
    mov byte ptr outbuf[di], al
    Inc di
    mov counto, di
    endm

outw macro reg, port
    mov di, counto
    mov word ptr outbuf[di], ax
    Inc di
    mov counto, di
    endm
```

将软件中的 I/O 指令用上述宏定义代替后的软件与原软件的功能是一样的，但是在性能上还是有所差别。由于这次测试主要是功能测试，所以仿真成立。

2) 软件插桩

为了便于测试完成后分析软件的执行路径，必须在软件中加入特定的输出语句。这样，我们才能在分析软件的输出数据时进行对比，得到软件的实际运行情况。可在遥控摄像头的控制及通信软件中加入下列语句：

```
MOV DADIAN, PATH[NO]
```

将软件中的上述插入点写入软件的输出结果中，测试结束后分析语句时，就可以根据路径编号来分析软件的执行路径。

3) 软件移植

由于遥控摄像头的控制及通信软件在实际环境中的执行方式是直接操作底层硬件，而 DOS 系统中的软件是和操作系统交互的，因此，为了完成测试，必须对软件进行移植，使

软件能够在 DOS 系统中运行。

在基于 8086 组建的嵌入式系统中，一般将软件安排在特定的存储地址中，系统启动时，CPU 指令首先指向 FF000H，在 FF000H 单元中安排特定的跳转语句，使软件跳转到存储器中存放软件的地址，然后系统进入正常运行。然而，在 DOS 操作系统中不能直接操作 CPU 指令指针，使 CPU 指令指针切换到存放特定软件的地址。DOS 系统下的可执行程序必须符合 DOS 系统的规范。我们可以在程序中插入驱动模块，然后对程序进行重新编译，使它符合 DOS 系统的规范，就可以解决上述问题。

至此，该软件的测试环境已经建立完成。被测软件、插桩模块、驱动模块和 DOS 系统及 PC 机就组成了遥控摄像头的控制及通信软件的测试环境。

9.7 思考与习题

1. 嵌入式软件测试的过程是什么？
2. 嵌入式软件的测试方法与普通软件测试有何不同？
3. 嵌入式软件的测试工具有哪些？

第 10 章　软件测试管理

作为软件项目开发的一个必要的组成部分，软件测试需要良好的组织和管理。本章从过程管理、需求管理、配置管理、缺陷管理和风险管理几个方面来谈谈测试管理问题。

10.1　过 程 管 理

软件测试过程管理在各个阶段的具体内容不同，在每个阶段，测试任务都要经过从计划、设计、执行到结果分析、总结等一系列相同步骤。

软件测试过程管理主要集中在软件测试项目启动、测试计划制定、测试用例设计、测试执行、测试结果审查和分析，以及如何开发或使用测试过程管理工具。

10.1.1　测试的组织

实施一个测试的首要步骤之一就是考虑测试中涉及到的人员的高级组织、他们之间的相互关系以及如何将测试过程集成到现有的业务管理结构中。

图 10.1 给出了测试过程组织的一种常见方法。

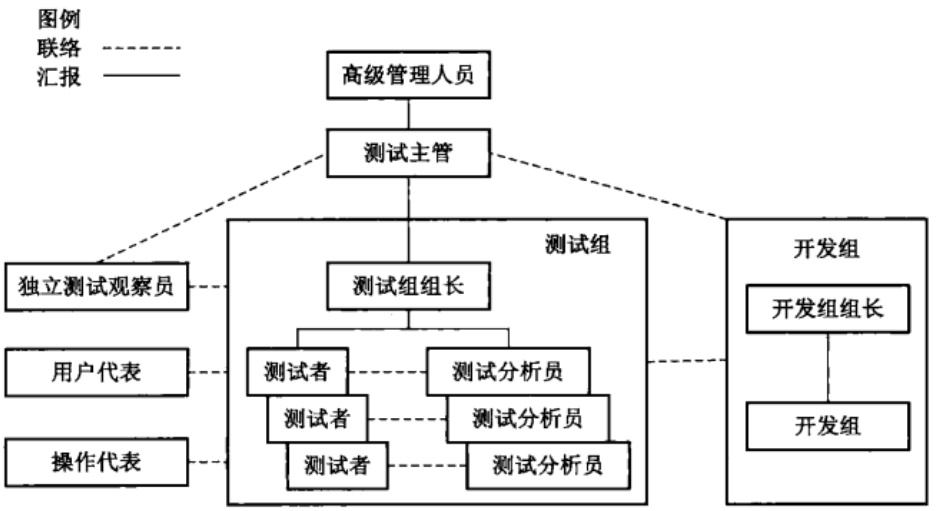


图 10.1　测试过程组织

1) 测试主管

测试主管有权管理测试过程日常的组织，负责保证在给定的时间、资源和费用的限制下设计测试项目产生满足所需的质量标准的产品。

测试主管在适当的时候也要负责与开发组进行联络，保证他们遵循在过程中引用的单元测试和集成测试方法。测试主管还要同独立测试观察员联系，接收有关没有正确遵循测试过程的测试项目的报告。

测试主管向公司内的高级主管或领导报告，如质量保证主管或信息技术领导。在大的公司中，尤其对于那些遵循规范的项目管理过程的公司中，测试主管可以向测试程序委员会报告，该委员会负责把握测试程序的项目管理的总体方向。

测试主管的职位可以是一个兼职的角色，可以由一个现有的高级主管或领导(如 QA 主管或 IT 领导)兼任。在测试涉及的人员相对较少的小公司中尤其可能出现这种情况。

2) 测试组组长

测试组组长有权运作一个项目。其职责包括为测试分析员和测试者分配任务，按照预定的计划监控他们的工作进度，建立和维护测试项目文件系统，保证产生测试项目相关材料顺利完成。这些材料包括：测试计划文档、测试规范说明文档。测试组组长负责完成这些文档，也可以授权测试分析员来完成这些文档。

测试组组长听取一个或多个测试分析员和测试者的报告，并向测试主管汇报。测试组组长将和独立测试观察员联系(如讨论参加一个特定测试的可行性)，适当的时候还会和开发组组长联系(完成测试项目的早期计划和基本的测试设计，并确定 AUT 测试的有效性)。在验收测试时，测试组组长负责和用户代表、操作代表联系，以便有一个或多个用户来执行用户和操作验收测试。

3) 测试分析员

测试分析员负责设计和实现用于完成 AUT 测试的一个或多个测试脚本以及相关的测试用例。测试分析员也可以派去协助测试组组长生成测试规格说明文档。

在调试测试用例的设计过程中，测试分析员需要分析 AUT 的需求规格说明，以便确定必须测试的特定需求。在这个过程中，测试分析员应该优先考虑测试用例，以反映被确认的特性的重要性以及在正常使用 AUT 中导致失败的特性的风险。可以采用给每个测试用例赋一个高、中或低的值的方法来完成这一工作。这是一项重要的工作，因为这种方法可以帮助测试组组长在时间和资源有限的情况下集中测试的人力。

在完成测试项目后，测试分析员负责备份和归档所有的测试文档和材料。这些材料将提交给测试组组长进行归档。测试分析员还负责完成一份测试总结报告，这份报告简要介绍该测试项目中的关键点。

测试分析员要向测试组组长汇报，并在开始测试 AUT 之前和测试者联系，向他们简单介绍测试者的任务。

4) 测试者

测试者主要负责执行由测试分析员建立的测试脚本，并负责解释测试用例结果并将结果记录到文档中。

在执行测试脚本之前，测试者首先要建立和初始化测试环境，其中包括测试数据和测试硬件，以及其它支持测试所需的软件(加模拟器和测试辅助程序)。

在测试执行过程中，测试者负责填写测试结果记录表格，以便记录执行每个测试脚本时观察到的结果。测试者使用测试脚本对预期结果进行描述。

在完成测试以后，测试者还负责备份测试数据、模拟器或测试辅助程序以及测试中使

用的硬件的说明。这些材料将提交给测试组组长归档。

10.1.2　测试计划阶段

测试计划要针对测试目的来规定测试的任务、所需的各种资源和投入、人员角色的安排、预见可能出现的问题和风险，以指导测试的执行，最终实现测试的目标，保证软件产品的质量。

编写测试计划的目的是：

(1) 为测试各项活动制定一个现实可行的、综合的计划，包括每项测试活动的对象、范围、方法、进度和预期结果。

(2) 为项目实施建立一个组织模型，并定义测试项目中每个角色的责任和工作内容。

(3) 开发有效的测试模型，能正确地验证正在开发的软件系统。

(4) 确定测试所需要的时间和资源，以保证其可获得性和有效性。

(5) 确立每个测试阶段测试完成以及测试成功的标准和要达到的目标。

(6) 识别出测试活动中的各种风险，并消除可能存在的风险，降低那些不可能消除的风险所带来的损失。

测试计划包括如下内容。

1．测试策略的制定

测试策略描述当前测试的目标和所采用的测试方法。这个目标不是上述测试计划的目标，而是针对某个应用软件系统或程序的。具体的测试任务要达到的预期结果，包括在规定的时间内哪些测试内容要完成，软件产品的特性或质量在哪些方面要得到确认。测试策略还要描述测试不同阶段(单元测试、集成测试、系统测试)的测试对象、范围和方法以及每个阶段内所要进行的测试类型(功能测试、性能测试、压力测试等)。在制定测试策略前，要确定测试策略项，测试策略包括：

(1) 要使用的测试技术和工具。如 60%用工具自动测试，40%手工测试。

(2) 测试完成标准。用以计划和实施测试，以及通报测试结果。如 95%测试用例通过并且重要级别的缺陷全部解决。

(3) 影响资源分配的特殊考虑。例如有些测试必须在周末进行，有些测试必须通过远程环境执行，有些测试需考虑与外部接口或硬件接口的连接。

(4) 在确认测试方法时，要根据实际情况，结合测试策略的特点来选择合适的方法。

(5) 根据是否需要执行被测软件来划分，测试可分为静态测试和动态测试。静态测试，如规格说明书、程序代码的审查，在工作中容易被忽视，在测试策略上应说明如何加强这些环节。

(6) 根据是否针对系统的内部结构和具体实现算法来划分，测试分为“白盒”测试和“黑盒”测试。如何将“白盒”测试和“黑盒”测试有机地结合起来，也是测试策略要处理的问题之一。

2．测试计划过程

测试计划经过计划初期、起草、讨论、审查等不同阶段，最终生成。测试计划过程如下所示：

(1) 计划初期是收集整体项目计划、需求分析、功能设计、系统原型、用例报告等文档或信息，理解用户的真正需求，了解技术难点、弱点或新的技术，和其余项目相关人员交流，在各个主要方面达到一致的理解。

(2) 测试计划最关键的一步就是确定测试需求、测试层次。将软件分解成单元，对各个单元写成测试需求，测试需求也是测试设计和开发测试用例的基础，是用来衡量测试覆盖率的重要指标。

(3) 计划起草。根据计划初期所掌握的各种信息、知识，确定测试策略，设计测试方法，完成测试计划的框架。

(4) 内部审查。在提供给其它部门讨论之前，先在测试小组或部门内部进行审查。

(5) 计划讨论和修改。召开有需求分析、设计、开发人员参加的计划讨论会议，测试组长将测试计划设计的思想、策略做较详细的介绍，并听取大家对测试计划中各个部分的意见，进行讨论交流。

(6) 测试计划的多方审查。项目中的每个人(即市场人员、开发人员、技术支持人员及测试人员)都应当参与审查。

(7) 测试计划的定稿和批准。在计划讨论、审查的基础上，综合各方面的意见，就可以完成测试计划书，然后报给测试经理或 QA 经理，得到批准，方可执行。

3．测试计划的要点

软件测试计划的内容主要包括：产品基本情况、测试需求说明、测试策略说明、测试资源配置、计划表、问题跟踪报告、测试计划的评审、结果等。除了产品基本情况、测试需求说明、测试策略等，测试计划的焦点集中在：

(1) 计划的目的：项目的范围和目标，各阶段的测试范围、技术约束和管理特点。

(2) 项目估算：使用的历史数据，使用的评估技术，工作量、成本、时间估算依据。

(3) 风险计划：测试可能存在的风险的分析、识别，以及风险的回避、监控、管理。

(4) 日程：项目工作分解结构，并采用时限图、甘特图等方法制定时间/资源表。

(5) 项目资源：人员、硬件和软件等资源的组织和分配，人力资源是重点，日程安排是核心。

(6) 跟踪和控制机制：质量保证和控制，变更管理和控制等。测试计划书的内容也可以按集成测试、系统测试、验收测试等阶段去组织，为每一个阶段制定一个计划书，也可以为每个测试任务/目的(安全性测试、性能测试、可靠性测试等)制定特别的计划书。

此外，可以对上述测试计划书的每项内容制定一个具体实施的计划，如对每个阶段的测试重点、范围、所采用的方法、测试用例设计的思想、提交的内容等进行细化，供测试项目组的内部成员使用。对于一些重要的项目，会形成一系列的计划书，如测试范围/风险分析报告、测试标准工作计划、资源和培训计划、风险管理计划、测试实施计划、质量保证计划等。

4．测试计划的编写内容

测试计划内容有被测软件的背景信息、测试目标、测试步骤、测试数据整理以及评估准则。它包括：

(1) 测试环境。在不同的条件下进行测试，所得到的结果是不同的。在测试计划中测试

环境指用户使用环境或业务运行环境。

(2) 测试基本原理和策略。

(3) 测试计划阶段划分。

(4) 测试计划要点。

(5) 功能描述和功能覆盖说明。

(6) 测试用例清单。说明每个测试用例所测试的内容。

(7) 测试开始准则和退出准则。

每个测试用例的序言至少包括下列信息：

(1) 测试用例说明和用途。

(2) 设置需求(输入、输出)。

(3) 运行测试用例的操作命令。

(4) 正常和异常信息。

(5) 编写测试用例的作者。

10.1.3　软件测试设计和开发

当测试计划完成之后，测试过程就要进入软件测试设计和开发阶段。软件测试设计是建立在测试计划书的基础上的。认真理解测试计划的测试大纲、测试内容及测试的通过准则，通过测试用例来完成测试内容与程序逻辑的转换，作为测试实施的依据，以实现所确定的测试目标。软件设计是将软件需求转换成为软件表示的过程，主要描绘出系统结构、详细的处理过程和数据库模式；软件测试设计则是将测试需求转换成测试用例的过程，它要描述测试环境、测试执行的范围、层次和用户的使用场景以及测试输入和预期的测试输出等。所以软件测试设计和开发是软件测试过程中一个技术深、要求高的关键阶段。

软件测试设计和开发的主要内容有：

(1) 制定测试的技术方案，确认各个测试阶段要采用的测试技术、测试环境和平台，以及选择什么样的测试工具。系统测试中的安全性、可靠性、稳定性、有效性等测试技术方案是这部分工作内容的重点。

(2) 设计测试用例。根据产品需求分析、系统设计等规格说明书，在测试技术选择的方案基础上，设计具体的测试用例。

(3) 设计测试用例特定的集合，满足一些特定的测试目的和任务，即根据测试目标、测试用例的特性和属性(优先级、层次、模块等)来选择不同的测试用例，构成执行某个特定测试任务的测试用例集合(组)，如基本测试用例组、专用测试用例组、性能测试用例组、其它测试用例组等。

(4) 测试开发。根据所选择的测试工具，将所有可以进行自动化测试的测试用例转换为测试脚本的过程。

(5) 测试环境的设计。根据所选择的测试平台以及测试用例所要求的特定环境，进行服务器、网络等测试环境的设计。

在软件测试设计和开发阶段，按国家标准 GB/T 9386—1988《计算机软件测试文件编制规范》的要求，编写“测试设计说明”、“测试用例说明”、“测试规程说明”、“测试项传递

报告”等文档。

1. 测试用例设计的方法和管理

软件测试用例设计的方法有“白盒”测试和“黑盒”测试相对应的设计方法。“黑盒”测试的用例设计，采用等价类划分、因果图法、边值分析、用户界面测试、配置测试、安装选项验证等方法，适用于功能测试和验收测试。“白盒”测试的用例设计有以下方法：

(1) 采用逻辑覆盖(包括程序代码的语句覆盖、条件覆盖、分支覆盖)的结构测试用例的设计方法。

(2) 基于程序结构的域测试用例设计方法。“域”是指程序的输入空间，域测试正是在分析输入空间的基础上，完成域的分类、定义和验证，从而对各种不同的域选择适当的测试点(用例)进行测试。

(3) 数据流测试用例设计的方法，是通过程序的控制流，从建立的数据目标状态的序列中发现异常的结构测试方法。

(4) 根据对象状态或等待状态变化来设计测试用例，也是比较常见的方法。

(5) 基于程序错误的变异来设计测试用例，可以有效地发现程序中某些特定的错误。

(6) 基于代数运算符号的测试用例设计方法，受分支问题、二义性问题和大程序问题的困扰，使用较少。

测试用例要经过创建、修改和不断改善的过程。一个测试用例具有以下属性：

(1) 测试用例的优先级次序。优先级越高，被执行的时间越早，执行的频率越高。由最高优先级的测试用例组来构成基本验证测试，每次构建软件包时，都要被执行一遍。

(2) 测试用例的目标性。有的测试用例是为主要功能而设计，有的测试用例是为次要功能而设计，有的为系统的负载而设计，有的则为一些特殊场合而设计。因此，需要根据不同的目标设计不同的测试用例。

(3) 测试用例所属的范围。测试用例属于哪一个组件或模块，这种属性被用来管理测试用例。

(4) 测试用例的关联性。测试用例一般是和软件产品特性相联系的，多数情况下验证某个产品的功能。这种属性可以被用于验证被修改的软件缺陷，或对软件产品紧急补丁包进行测试。

(5) 测试用例的阶段性。测试用例处于单元测试、集成测试、系统测试、验收测试中的某一个阶段。这样对每个阶段，构造一个测试用例的集合被执行，并容易计算出该阶段的测试覆盖率。

(6) 测试用例的状态性。当前是否有效，如果无效，被置于 Inactive 状态，不会被运行；只有被激活的(active)测试用例才被运行。

(7) 测试用例的时效性。针对同种功能，可能所用的测试用例不同，这是由于不同的产品版本在产品功能、特性等方面的要求不同。

(8) 所有者、日期等特性。测试用例还包括由谁、在什么时间创建，又由谁、在什么时间修改。

根据上述特性，再结合测试用例的编号、标题、描述(条件、步骤、期望结果)等，就可以对测试用例进行基于数据库方式的良好管理。测试用例设计完之后，要经过非正式和正

式的审查。

审查完的测试用例经修改后，就可以直接用于手工测试或用于测试脚本的开发。

2. 测试开发

根据所选择的测试工具脚本语言，编写测试脚本，将所有可以进行自动化测试的测试用例转化为测试脚本。其输入就是基于测试需求的测试用例，输出是测试脚本和与之相对应的期望结果，这种期望结果一般存储在数据库中或特定的格式化文件中。

测试开发的步骤如下：

首先要设立测试脚本开发环境，安装测试工具软件，设置管理服务器和具有代理的客户端，建立项目的共享路径、目录，并能连接到脚本存储库和被测软件等。

然后执行录制测试的初始化过程、独立模块过程、导航过程和其它操作过程。结合已经建立的测试用例，将录制的测试脚本进行组织、调试和修改，构造成一个有效的测试脚本体系，并建立外部数据集合。

10.1.4　测试执行阶段

当测试用例的设计和测试脚本的开发完成之后，就开始执行测试。测试的执行有手工测试和自动化测试两种。手工测试指在合适的测试环境上，按照测试用例的条件、步骤要求，准备测试数据，对系统进行操作，比较实际结果和测试用例所描述的期望结果，以确定系统是否正常运行或正常表现。自动化测试是通过测试工具，运行测试脚本，并自动记录下测试结果。

针对每个测试阶段(代码审查、单元测试、集成测试、功能测试、系统测试、验收测试和安装测试等)的结果进行分析，保证每个阶段的测试任务得到执行，达到阶段性目标。

测试过程中发现的软件错误或缺陷，可提交或纳入到软件缺陷管理过程中。bug 的跟踪和管理一般由数据库系统来执行，依赖于如下的规则和流程进行的。

- 设计好每个 bug 应该包含的信息条目、状态分类等。
- 通过系统自动发出邮件给相应的开发人员和测试人员，使得任何 bug 都不会错过，并能得到及时处理。
- 通过日报、周报等各类项目报告来跟踪目前 bug 状态。
- 在各个大小里程碑之前，召开有关人员的会议，对 bug 进行会审。
- 通过一些历史曲线、统计曲线等进行分析，预测未来情况。

10.1.5　测试执行结束和测试总结

测试执行全部完成，并不意味着测试项目的结束。测试项目结束的阶段性标志是将测试报告或质量报告发出去后，得到测试经理或项目经理的认可。除了测试报告或质量报告的写作之外，还要对测试计划、测试设计和测试执行等进行检查、分析，完成项目的总结，编写“测试总结报告”。通常包括以下活动：

(1) 审查测试全过程：在原来跟踪的基础上，要对测试项目进行全过程、全方位的审视，检查测试计划、测试用例是否得到执行，检查测试是否有漏洞。

(2) 对当前状态的审查：包括产品 bug 和过程中没有解决的各类问题。对产品目前存

在的缺陷进行逐个的分析，了解对产品质量影响的程度，从而决定产品的测试能否告一段落。

(3) 结束标志：根据上述两项的审查进行评估，如果所有测试内容完成、测试的覆盖率达到要求以及产品质量达到已定义的标准，就可以对测试报告定稿并发送出去。

(4) 项目总结：通过对项目中的问题分析，找出流程、技术或管理中所存在的问题根源，避免今后发生类似错误，并获得项目成功经验。

10.1.6 测试过程改进

软件测试过程改进主要着眼于合理调整各项测试活动的时序关系，优化各项测试活动的资源配置以及实现各项测试活动效果的最优化。在软件测试过程中，过程改进被赋予了举足轻重的地位，在测试计划、实施、检查、改进的循环中，过程改进既是一次质量活动的终点，又是下次质量活动的原点，起着承上启下的作用。

1. 软件测试过程改进的概念

测试过程的改进对象应该包括三个方面：组织、技术和人员。测试过程改进需要对组织给予特别关注，因为过程都是基于特定的组织架构建设的，而且组织设置是否合理对过程的好坏有决定性的影响。软件测试组织的不良架构通常表现在：

(1) 没有恰当的角色追踪项目进展。

(2) 没有恰当的角色进行缺陷控制、变更和版本追踪。

(3) 项目在测试阶段效率低下、过程混乱。

(4) 只有测试经理了解项目，项目成了个人的项目，而不是组织的项目。

(5) 关心进度，而忘记了项目的另外两个要素——质量和成本。

上述问题可从组织上找出原因。因此在测试过程改进中可以先将测试从开发活动中分离出来，把缺陷控制、版本管理和变更管理从项目管理中分离出来。此外，需要给测试经理赋予明确的职责和目标。技术的改进包括对流程、方法和工具的改进，它包括组织或者项目对流程进行明确的定义，引入统一的管理方法，并使用标准的经过组织认可的工具和模板。人员的改进主要是指对企业文化的改进，它将促使建立高效率的团队和组织。

由于测试过程改进是一项长期的、没有终点的活动，而且要获得改进过程的收益也是长期的过程，因此在起步实施测试过程改进时，要充分考虑战略，并根据公司的战略目标确定测试部门的战略，描绘远景。将测试过程改进与公司战略目标相联系，是改进成功实施的必要条件，也是各公司在实施测试过程改进中获得的最佳实践。在研究过程中，组织的规划内容通常包括：

(1) 绘制远景。如提升管理成熟度，提高测试生产率，促使部门测试能力达到公司领先水平。

(2) 战略分析。如在部门内制订三年计划。以内部人员为主，引入适当的培训，进行相关过程域改进并达到 CMMI3 成熟度，适时进行评估，最终目标为 CMMI4。

(3) 优缺点评估。上述战略方法的优点在于前期以内部改进为宗旨，使过程改进更符合组织的实际情况。但缺点是不以正式评估作为目标，过程改进的动力不足。

在改进的不同时期和阶段，选择的策略也不同，组织应根据实际情况进行选择。下面

列举了一些可供参照的主要策略方法：

(1) 重诊断，轻评估。要以诊断和解决测试过程中的实际问题作为测试过程改进的目的，不能盲目追求商业评估。

(2) 实施制度化的同时建设企业文化。实施全面制度化的管理是过程改进的有效保障，制度和组织文化总是互相依存的。没有良好的文化保障，制度化将困难重重；而没有制度的支撑，文化也将是无本之木。

(3) 引入软件工具。推行配置、自动化测试和缺陷跟踪等工具，将有效地分解事务性工作，可以缓解人力资源不足的困难。常见的过程管理方面的工具包括 Rational 公司的 ClearCase、ClearQuest，CA 公司的 CCC/Harvest 等。

(4) 建设管理和工程基础。为了解决基础薄弱的问题，需要在测试过程改进前期为相关部门和员工进行基础管理和基本软件工程的课程培训。

(5) 发动全员参与。全员参与可以分三个层面来理解：第一，站在高于项目管理的层面；第二，站在项目管理的层面；第三，站在开发人员和测试人员的层面。充分调动各方面人员的积极性。

(6) 现有过程的复用。该原则可以充分利用现有过程的合理部分，提高被改进过程的可接受程度和使用价值。

2. 软件测试过程改进的具体方法

过程改进在软件测试过程中占有举足轻重的位置，因此为了更好地保证软件质量，测试过程改进是测试人员经常要做的事情。下面列出了一些软件测试过程改进的具体方法：

(1) 调整测试活动的时序关系。在软件测试过程的测试计划中，不恰当的测试时序会引起误工和测试进度失控。例如，具体到某个工程实践中，有些测试活动是可以并行的，有些测试活动是可以归并完成的，有些测试活动在时间上存在线性关系等。所有这些一定要区分清楚并且要做最优化调整，否则会对测试进度产生不必要的影响。

(2) 优化测试活动资源配置。在软件测试过程中，必然会涉及到人力、设备、场地、软件环境与经费等资源。那么如何合理地调配各项资源给相关的测试活动是非常值得斟酌的，否则会引起误工和测试进度的失控。在测试资源配置中最常见的是人力资源的调配，测试部门如果能深入了解员工的专长与兴趣所在，在进行人员分配时，根据各自的特点进行分配，就能对测试活动的开展起到事半功倍的效果。

(3) 提高测试计划的指导性。测试计划的指导性是指测试计划的执行能力。在软件测试过程中，很多时候实际的测试和测试计划是脱节的，或者说很大程度上没有按照测试计划去执行。测试计划的完成不仅仅是起草测试大纲，而且是为了确保测试大纲中计划的内容能真正被执行、真正用于指导测试工作，为了更好地完成测试活动，保证软件的质量。

(4) 确立合理的度量模型和标准。在测试过程改进中，测试过程改进小组应根据企业与项目的实际情况制定适合自己公司的质量度量模型和标准，做出符合自己公司发展策略的投入。但是质量度量模型和标准的确立不是马上就可以进行的，而是测试过程改进小组随着测试过程的进行不断实践、不断总结、不断改进的。

(5) 提高覆盖率。覆盖率越高，表明测试的质量越高。覆盖率包括内容覆盖和技术覆盖。内容覆盖指的是起草测试计划、设计测试用例、执行测试用例和跟踪软件缺陷。内容覆盖

率越高，就越能避免故障被遗漏的情况。技术覆盖指一项技术指标要尽可能地做到测试技术的覆盖，采用科学的方法来验证某项指标，可以更好地保证产品的质量。

10.2 需求管理

10.2.1 需求管理概述

Standish Group 从 1994 年到 2001 年的 CHAOS Reports 证实，导致项目失败的最重要的原因与需求有关。2001 年，Standish Group 的 CHAOS Reports 报导了该公司的一项研究，该公司对多个项目作调查后发现，百分之七十四的项目是失败的，即这些项目不能按时按预算完成。其中提到最多的导致项目失败的原因就是“变更用户需求”。

因此，良好需求管理的第一步就是对什么是需求管理达成共识。Rational 把需求定义为“(正在构建的)系统必须符合的条件或具备的功能”。电气和电子工程师学会使用的定义与此类似。著名的需求工程设计师 Merlin Dorfman 和 Richard H.Thayer 提出了一个包容且更为精练的定义，它特指软件方面但不仅仅限于软件：软件需求可定义为用户解决某一问题或达到某一目标所需的软件功能。系统或系统构件为了满足合同、规约、标准或其它正式实行的文档而必须满足或具备的软件功能。

需求管理就是一种获取、组织并记录系统需求的系统化方案，以及一个使客户与项目团队对不断变更的系统需求达成并保持一致的过程。

这个定义与 Dorfman 和 Thayer 以及 IEEE 的“软件需求工程”的定义相似。需求工程包括获取、分析、规定、验证和管理软件需求，而“软件需求管理”则是对所有相关活动的规划和控制。这里介绍的以及 IBM Rational 提出的需求管理定义包括了所有这些活动。它们的区别主要在于这里选用了“管理”这个词，而不是“工程”。管理这个词更适合用来描述所有涉及到的活动，并且它准确地强调了追踪变更以保持涉众与项目团队之间共识的重要性。

10.2.2 软件测试中的需求分析

1．熟悉需求背景及商业目标

(1) 了解清楚项目发起的原因是为了解决用户的什么问题。

(2) 清楚当前的解决方案是不是最优的以及为什么会这样做。

2．业务模型法

(1) 考虑本项目与外部系统的交互，划分系统边界(除了本项目的需求中要求做的事情，其它的都可以是外部系统，本系统和外部系统之间的交互就是系统的边界)。可以参考系统分析说明书。

(2) 确定测试范围和关注点。系统的边界是测试的重点，特别需要关注边界交互时的数据交互。

3．业务场景法

(1) 考虑用例的调用者，考虑每一个用例提供的服务是供哪些外部用例或者系统调用，

找出所有的调用者。调用的前提、约束都要考虑。每一个调用都可以考虑成一个大的业务流程。(一般和外部有交互的用例出错的概率比较大，需要重点关注。具体被哪些外部调用，每个产品线都需要自己整理添加。)

(2) 考虑系统内部各个用例之间的交互(有可能 PD 划分用例的粒度不同，我们暂时考虑用户一次提交并且系统的状态及数据发生变化的功能是一个用例)，形成内部业务流程图。需要分析每个用例之间的约束关系、执行条件，设计出各种业务流程图。

4．功能分解法

(1) 业务功能：与用户实际业务直接相关的功能或细节。

(2) 辅助功能：辅助完成业务功能的一些功能或细节，比如设置过滤条件。

(3) 数据约束：主要用于控制在执行功能时，数据的显示范围、数据之间的关系等。

(4) 易用性需求：产品中必须提供便于功能操作使用的一些细节，比如快捷键就是典型的易用性需求。

(5) 编辑约束：在功能执行时，对输入数据项目设定一些约束性条件，比如只能输入数字。

(6) 参数需求：在功能中，需要根据参数设置不同，进行不同处理的细节。

(7) 权限需求：这里的权限是指在功能的执行过程中，根据不同的权限进行不同的处理，不包括直接限制某个功能的权限。

(8) 性能约束：执行功能时，必须满足的性能要求目前基本不涉及。

10.3　软件配置管理

10.3.1　软件配置管理概述

软件配置管理(Software Configuration Management，SCM)是一种标识、组织和控制修改的技术。软件配置管理应用于整个软件工程过程。

软件配置管理是在贯穿整个软件生命周期中建立和维护项目产品的完整性。它的基本目标包括：

(1) 软件配置管理的各项工作是有计划进行的。

(2) 被选择的项目产品得到识别、控制并且可以被相关人员获取。

(3) 已识别出的项目产品的更改得到控制。

(4) 使相关组别和个人及时了解软件基准的状态和内容。

为了达到上述目标，如下的方针应该得到贯彻执行：

(1) 技术部门经理和具体项目主管应该使用和遵循 XSSC 的 OSSP 中所描述的软件配置管理的工作过程。

(2) 施行软件配置管理的职责应被明确分配。相关人员得到软件配置管理方面的培训。

(3) 技术部门经理和具体项目主管应该明确他们在相关项目中所担负的软件配置管理方面的责任。

(4) 软件配置管理工作应该享有足够的资金支持，这需要在客户、技术部门经理和具体

项目主管之间协商。

(5) 软件配置管理应该实施于如下产品：对外交付的软件产品以及那些被选定的在项目中使用的支持类工具等。

(6) 软件配置的整体性在整个项目生命周期中得到控制。

(7) 软件质量保证人员应该定期审核各类软件基准以及软件配置管理工作。

(8) 软件基准的状态和内容能够及时通知给相关组别和个人。

10.3.2 软件配置管理角色职责

对于任何一个管理流程来说，保证该流程正常运转的前提条件就是要有明确的角色、职责和权限的定义。特别是在引入了软件配置管理的工具之后，比较理想的状态就是：组织内的所有人员按照不同的角色要求，根据系统赋予的权限来执行相应的动作。因此，在本文所介绍的这个软件配置管理过程中主要涉及下列的角色和分工。

1) 项目经理

项目经理(Project Manager，PM)是整个软件研发活动的负责人，他根据软件配置控制委员会的建议，批准配置管理的各项活动并控制它们的进程。其具体职责为以下几项：

(1) 制定和修改项目的组织结构和配置管理策略。

(2) 批准、发布配置管理计划。

(3) 决定项目起始基线和开发里程碑。

(4) 接受并审阅配置控制委员会的报告。

2) 配置控制委员会

(1) 配置控制委员会(Configuration Control Board，CCB)负责指导和控制配置管理的各项具体活动的进行，为项目经理的决策提供建议。其具体职责为以下几项：定制开发子系统；定制访问控制；制定常用策略。

(2) 建立、更改基线的设置，审核变更申请。

(3) 根据配置管理员的报告决定相应的对策。

3) 配置管理员

配置管理员(Configuration Management Officer，CMO)根据配置管理计划执行各项管理任务，定期向 CCB 提交报告并列席 CCB 的例会。其具体职责为以下几项：

(1) 软件配置管理工具的日常管理与维护。

(2) 提交配置管理计划。

(3) 各配置项的管理与维护。

(4) 执行版本控制和变更控制方案。

(5) 完成配置审计并提交报告。

(6) 对开发人员进行相关的培训。

(7) 识别软件开发过程中存在的问题并拟就解决方案。

4) 系统集成员

系统集成员(System Integration Officer，SIO)负责生成和管理项目的内部和外部发布版本，其具体职责为以下几项：

(1) 集成修改。

(2) 构建系统。

(3) 完成对版本的日常维护。

(4) 建立外部发布版本。

5) 开发人员

开发人员(Developer，DEV)的职责就是根据组织内确定的软件配置管理计划和相关规定，按照软件配置管理工具的使用模型来完成开发任务。

10.3.3 软件配置管理过程描述

一个软件研发项目一般可以划分为三个阶段：计划阶段、开发阶段和维护阶段。然而从软件配置管理的角度来看，后两个阶段所涉及的活动是一致的，所以就把它们合二为一，称为“项目开发和维护”阶段。

1．项目计划阶段

一个项目设立之初，PM 首先需要制定整个项目的计划，它是项目研发工作的基础。在有了总体研发计划之后，软件配置管理的活动就可以开展了。

在软件配置管理计划的制定过程中，它的主要流程如下：

(1) CCB 根据项目的开发计划确定各个里程碑和开发策略。

(2) CMO 根据 CCB 的规划，制定详细的配置管理计划，交 CCB 审核。

(3) CCB 通过配置管理计划后交项目经理批准，发布实施。

2．项目开发和维护阶段

这一阶段是项目研发的主要阶段。在这一阶段中，软件配置管理活动主要分为三个层面：

(1) 主要由 CMO 完成的管理和维护工作。

(2) 由 SIO 和 DEV 具体执行软件配置管理策略。

(3) 变更流程。

这三个层面是彼此之间既独立又互相联系的有机的整体。

在这个软件配置管理过程中，它的核心流程应该是这样的：

(1) CCB 设定研发活动的初始基线。

(2) CMO 根据软件配置管理规划设立配置库和工作空间，为执行软件配置管理做好准备。

(3) 开发人员按照统一的软件配置管理策略，根据获得的授权的资源进行项目的研发工作。

(4) SIO 按照项目的进度集成组内开发人员的工作成果，并构建系统，推进版本的演进。

(5) CCB 根据项目的进展情况，审核各种变更请求，并适时地划定新的基线，保证开发和维护工作的有序进行。

这个流程就是如此循环往复，直到项目的结束。当然，在上述的核心过程之外，还涉及其它一些相关的活动和操作流程，下面按不同的角色分工予以列出，如图 10.2 所示。

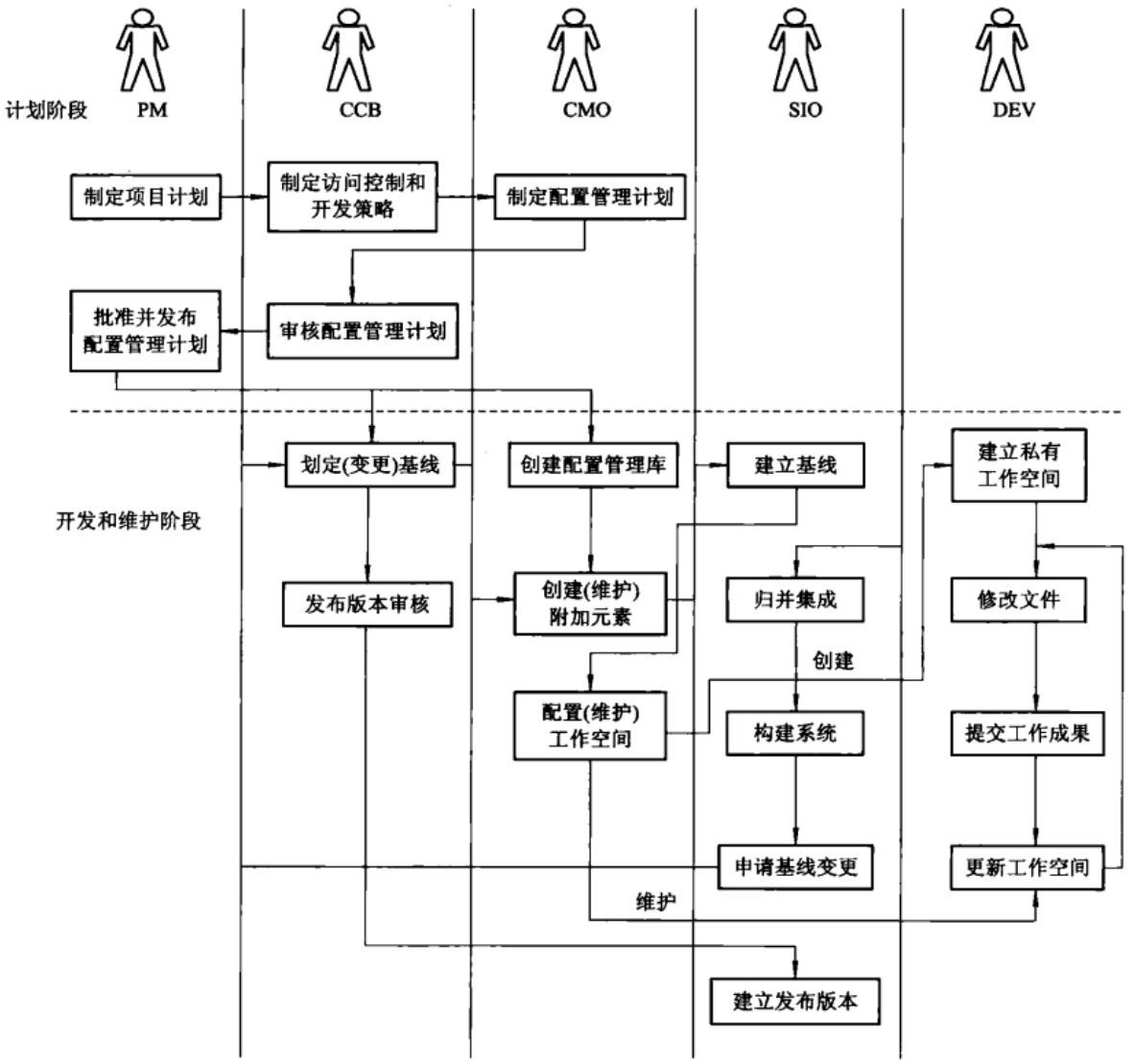


图 10.2 软件配置管理基本流程

10.3.4 软件配置管理的关键活动

1. 配置项(Software Configuration Item，SCI)识别

Pressman 对于 SCI 给出了一个比较简单的定义："软件过程的输出信息可以分为三个主要类别：① 计算机程序(源代码和可执行程序)。② 描述计算机程序的文档(针对技术开发者和用户)。③ 数据(包含在程序内部或外部)。这些项包含了所有在软件过程中产生的信息，总称为软件配置项。"

软件的开发过程是一个不断变化着的过程，为了在不严重阻碍合理变化的情况下控制变化，软件配置管理引入了"基线(Base Line)"这一概念。IEEE 对基线的定义是这样的："已经正式通过复审核批准的某规约或产品，它因此可作为进一步开发的基础，并且只能通过正式的变化控制过程改变。"

软件的开发流程中把所有需加以控制的配置项分为基线配置项和非基线配置项两类。例如：基线配置项可能包括所有的设计文档和源程序等；非基线配置项可能包括项目的各类计划和报告等。所有配置项的操作权限应由 CMO 严格管理，基本原则是：基线配置项向软件开发人员开放读取的权限；非基线配置项向 PM、CCB 及相关人员开放。

2．工作空间管理

在引入了软件配置管理工具(如 CVS、VSS 等)之后，所有开发人员都会被要求把工作成果存放到由软件配置管理工具所管理的配置库中，或是直接工作在软件配置管理工具提供的环境之下。所以为了让每个开发人员和各个开发团队能更好地分工合作，同时又互不干扰，对工作空间的管理和维护也成为了软件配置管理的一个重要的活动。

一般来说，比较理想的情况是把整个配置库视为一个统一的工作空间，然后再根据需要把它划分为个人(私有)、团队(集成)和全组(公共)这三类工作空间(分支)，从而更好地支持将来可能出现的并行开发的需求。

每个开发人员按照任务的要求，在不同的开发阶段，工作在不同的工作空间上。例如：对于私有开发空间而言，开发人员根据任务分工获得对相应配置项的操作许可之后，即在自己的私有开发分支上工作，其所有工作成果体现为在该配置项的私有分支上的版本的推进，除该开发人员外，其他人员均无权操作该私有空间中的元素；而集成分支对应的是开发团队的公共空间，该开发团队拥有对该集成分支的读写权限，其他成员只有只读权限，它的管理工作由 SIO 负责；至于公共工作空间，则是用于统一存放各个开发团队的阶段性工作成果，它提供全组统一的标准版本，并作为整个组织的知识库。

3．版本控制

版本控制是软件配置管理的核心功能。所有置于配置库中的元素都应自动予以版本的标识，并保证版本命名的唯一性。版本在生成过程中，自动依照设定的使用模型自动分支、演进。除了系统自动记录的版本信息以外，为了配合软件开发流程的各个阶段，还需要定义、收集一些元数据来记录版本的辅助信息和规范开发流程，并对软件过程的度量做好准备。

4．变更控制

基线与变更控制紧密相连，在对各个 SCI 做出了识别，并且利用工具对它们进行了版本管理之后，如何保证它们在复杂多变的开发过程中真正地处于受控的状态，并在任何情况下都能迅速地恢复到任一历史状态就成为了软件配置管理的另一重要任务。因此，变更控制就是通过结合人的规程和自动化工具，以提供一个变化控制的机制。

变更控制的对象主要指配置库中的各基线配置项，变更管理的一般流程如下：

(1) (获得)提出变更请求。

(2) 由 CCB 审核并决定是否批准。

(3) (被接受)修改请求分配人员，提取 SCI，进行修改。

(4) 复审变化。

(5) 提交修改后的 SCI。

(6) 建立测试基线并测试。

(7) 重建软件的适当版本。

(8) 复审(审计)所有 SCI 的变化。

(9) 发布新版本。

在这样的流程中，CMO 通过软件配置管理工具来进行访问控制和同步控制，而这两种控制则是建立在前文所描述的版本控制和分支策略的基础上的。

5．配置状态报告

配置状态报告就是根据配置项操作数据库中的记录来向管理者报告软件开发活动的进展情况。这样的报告应该是定期进行，并尽量通过 CASE 工具自动生成，用数据库中的客观数据来真实反映各配置项的情况。

配置状态报告应根据报告着重反映当前基线配置项的状态，以作为对开发进度报告的参照。同时也能从中根据开发人员对配置项的操作记录来对开发团队的工作关系作一定的分析。

配置状态报告应该包括下列主要内容：

(1) 配置库结构和相关说明。

(2) 开发起始基线的构成。

(3) 当前基线位置及状态。

(4) 各基线配置项集成分支的情况。

(5) 各私有开发分支类型的分布情况。

(6) 关键元素的版本演进记录。

(7) 其它应予报告的事项。

6．配置审计

配置审计的主要作用是作为变更控制的补充手段，来确保某一变更需求已被切实实现。在某些情况下，它被作为正式的技术复审的一部分，但当软件配置管理是一个正式的活动时，该活动由 SQA 人员单独执行。

总之，软件配置管理的对象是软件研发活动中的全部开发资产。所有这一切都应作为配置项纳入管理计划统一进行管理，从而能够保证及时地对所有软件开发资源进行维护和集成。因此，软件配置管理的主要任务也就归结为以下几条：

(1) 制定项目的配置计划。

(2) 对配置项进行标识。

(3) 对配置项进行版本控制。

(4) 对配置项进行变更控制。

(5) 定期进行配置审计。

(6) 向相关人员报告配置的状态。

10.4 缺 陷 管 理

10.4.1 缺陷跟踪管理系统概述

缺陷跟踪管理系统(Defect Tracking System)是用于集中管理软件测试过程中发现的缺陷

的数据库程序，可以通过添加、修改、排序、查寻、存储操作来管理软件缺陷。

1) 缺陷跟踪管理系统的作用

(1) 便于缺陷的查找和跟踪。对于大中型软件的测试过程而言，报告的缺陷总数可能多达成千上万个，如果没有缺陷跟踪管理系统的支持，要求查找某个错误，其难度和效率可想而知。

(2) 便于协同工作。缺陷跟踪管理系统可以作为测试人员、开发人员、项目负责人、缺陷评审人员协同工作的平台。

(3) 保证测试工作的有效性。避免测试人员重复报错，同时也便于及时掌握各缺陷的当前状态，进而完成对应状态的测试工作。

(4) 便于跟踪和监控错误的处理过程和方法。可以方便地检查处理方法是否正确，跟踪处理者的姓名和处理时间，作为工作量的统计和业绩考核的参考。

2) 缺陷跟踪管理系统的实现原理

缺陷跟踪管理系统在实现技术层面上看是一个数据库应用程序。它包括前台用户界面、后台缺陷数据库以及中间数据处理层。

这类系统的用户界面所显示的信息一般应根据用户的角色不同(测试人员、开发人员、项目负责人、缺陷评审员等)而略有差异，因为各个角色使用该系统完成的任务各不相同，如测试人员用于报告缺陷或确认缺陷是否可以关闭，开发人员用于了解哪些缺陷需要他去处理以及缺陷经过处理后是否被关闭，而项目负责人需要及时了解当前有哪些新的缺陷，哪些必须及时修正等等。另外，不同角色所拥有的数据操作权限也不尽相同。例如开发人员无权通过其用户界面往数据库中填写新的缺陷信息，也无权关闭某个已知缺陷；而测试人员无权决定分配谁去修正某个已知缺陷，无权决定是否要修正某个缺陷。

10.4.2　软件缺陷内容

软件缺陷包括如下内容：

(1) 可追踪信息。如给每个缺陷分配一个缺陷编号，则每个编号必须是唯一的，可以根据该编号搜索、查看该缺陷的处理情况。

(2) 缺陷的基本信息。通常缺陷的基本信息包括缺陷状态、缺陷标题、缺陷严重程度、缺陷紧急程度、缺陷提交人、缺陷提交日期、缺陷所属、缺陷解决人、缺陷解决时间、缺陷解决结果、缺陷处理人、缺陷处理最终时间、缺陷处理结果、缺陷确认人、缺陷确认时间、缺陷确认结果等等。

(3) 缺陷的详细描述，即对缺陷的特征应做详细的描述。例如程序代码中的错误，应详细描述错误发生的软硬件环境、相关输入输出数据、出错时程序的状态等等，以方便编码人员进行错误复现和错误定位。又如设计规格说明中的错误，应指明它与高层软件开发文档(如需求规格说明)中哪些条款相违背，为什么判为违背，都需要描述清楚，以方便设计人员进一步核实。

(4) 必要的附件。有时，某些缺陷可能无法用语言文字表达清楚，如用户界面出现的一些缺陷，光用语言文字难以表述清楚。这时，就需要借助于屏幕拷贝等方式来进一步描述缺陷。

10.4.3 软件跟踪缺陷处理的一般流程

软件跟踪管理确保每个被发现的缺陷都能被解决。解决可能是指缺陷被修正，也可能是指项目组成员达成一致的处理意见(如不作处理)。

软件跟踪缺陷处理的流程如图 10.3 所示。

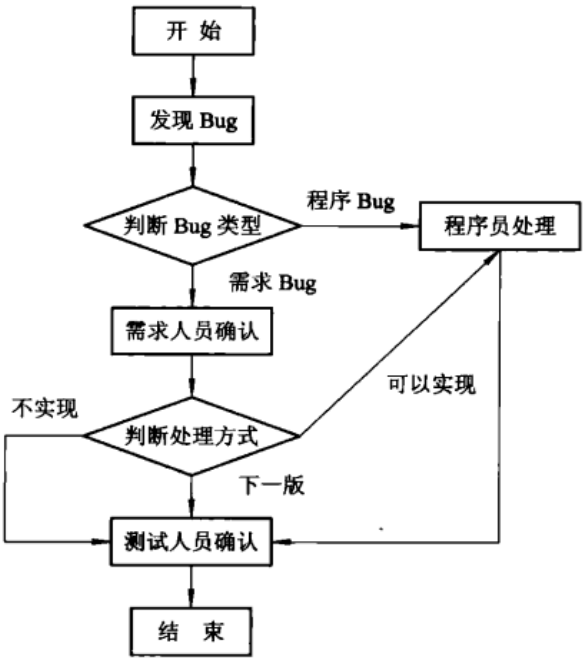


图 10.3 缺陷处理流程

10.5 风 险 管 理

软件项目风险是指在软件开发过程中遇到的预算和进度等方面的问题以及这些问题对软件项目的影响。软件项目风险会影响项目计划的实现，如果项目风险变成现实，就有可能影响项目的进度，增加项目的成本，甚至使软件项目不能实现。如果对项目进行风险管理，就可以最大限度地减少风险的发生。

软件项目的风险管理是软件项目管理的重要内容。在进行软件项目风险管理时，要辨识风险，评估风险出现的概率及产生的影响，建立一个规划来管理风险。

10.5.1 风险管理概述

近几年来软件开发技术、工具都有了很大的进步，但是软件项目开发超时、超支，甚至不能满足用户需求而根本没有得到实际使用的情况仍然比比皆是。软件项目开发和管理中一直存在着种种不确定性，严重影响着项目的顺利完成和提交。但这些软件风险并未得到充分的重视和系统的研究。直到 20 世纪 80 年代，Boehm 比较详细地对软件开发中的风险进行了论述，并提出软件风险管理的方法。Boehm 认为，软件风险管理指的是“试图以

一种可行的原则和实践，规范化地控制影响项目成功的风险”，其目的是“辨识、描述和消除风险因素，以免它们威胁软件的成功运作”。

10.5.2　软件项目风险管理

风险管理涉及的主要过程包括风险识别、风险量化、风险应对计划制定和风险监控，如图 10.4 所示。风险识别在项目的开始时就要进行，并在项目执行中不断进行。就是说，在项目的整个生命周期内，风险识别是一个连续的过程。

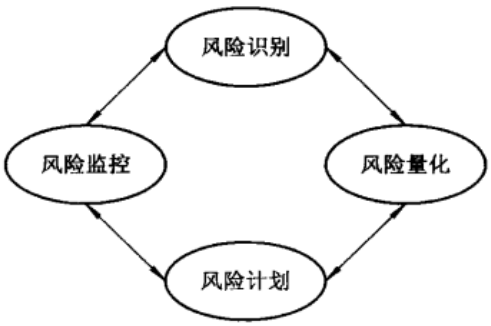


图 10.4　风险管理过程

风险识别：风险识别包括确定风险的来源、风险产生的条件，描述其风险特征和确定哪些风险事件有可能影响本项目。风险识别不是一次就可以完成的事，应当在项目的自始至终定期进行。

风险量化：涉及对风险及风险的相互作用的评估，是衡量风险概率和风险对项目目标影响程度的过程。风险量化的基本内容是确定哪些事件需要制定应对措施。

风险应对计划制定：针对风险量化的结果，为降低项目风险的负面效应制定风险应对策略和技术手段的过程。风险应对计划依据风险管理计划、风险排序、风险认知等依据，得出风险应对计划、剩余风险、次要风险以及为其它过程提供依据。

风险监控：涉及整个项目管理过程中的风险应对。该过程的输出包括应对风险的纠正措施以及风险管理计划的更新。

每个步骤所使用的工具和方法详见表 10.1。

表 10.1　风险管理过程中所使用的工具、方法

风险管理步骤	所使用的工具、方法
风险识别	头脑风暴法、面谈、Delphi 法、核对表、SWOT 技术
风险量化	风险因子计算、PERT 估计、决策树分析、风险模拟
风险应对计划制定	回避、转移、缓和、接受
风险监控	核对表、定期项目评估、挣值分析

10.5.3　软件项目中的风险

软件项目的风险无非体现在以下四个方面：需求、技术、成本和进度。IT 项目开发中常见的风险有如下几类：

1) 需求风险

(1) 需求已经成为项目基准，但需求还在继续变化；

(2) 需求定义欠佳，而进一步的定义会扩展项目范畴；

(3) 添加额外的需求；

(4) 产品定义含混的部分比预期需要更多的时间；

(5) 在做需求时客户参与不够；

(6) 缺少有效的需求变化管理过程。

2) 计划编制风险

(1) 计划、资源和产品定义全凭客户或上层领导口头指令，并且不完全一致；

(2) 计划是优化的，是“最佳状态”，但计划不现实，只能算是“期望状态”；

(3) 计划基于使用特定的小组成员，而那个特定的小组成员其实指望不上；

(4) 产品规模(代码行数、功能点、与前一产品规模的百分比)比估计的要大；

(5) 完成目标日期提前，但没有相应地调整产品范围或可用资源；

(6) 涉足不熟悉的产品领域，花费在设计和实现上的时间比预期的要多。

3) 组织和管理风险

(1) 仅由管理层或市场人员进行技术决策，导致计划进度缓慢，计划时间延长；

(2) 低效的项目组结构降低生产率；

(3) 管理层审查决策的周期比预期的时间长；

(4) 预算削减，打乱项目计划；

(5) 管理层作出了打击项目组积极性的决定；

(6) 缺乏必要的规范，导致工作失误与重复工作；

(7) 非技术的第三方的工作(预算批准、设备采购批准、法律方面的审查、安全保证等)时间比预期的延长。

4) 人员风险

(1) 作为先决条件的任务(如培训及其它项目)不能按时完成；

(2) 开发人员和管理层之间关系不佳，导致决策缓慢，影响全局；

(3) 缺乏激励措施，士气低下，降低了生产能力；

(4) 某些人员需要更多的时间适应还不熟悉的软件工具和环境；

(5) 项目后期加入新的开发人员，需进行培训并逐渐与现有成员沟通，从而使现有成员的工作效率降低；

(6) 由于项目组成员之间发生冲突，导致沟通不畅、设计欠佳、接口出现错误和额外的重复工作；

(7) 不适应工作的成员没有调离项目组，影响了项目组其他成员的积极性；

(8) 没有找到项目急需的具有特定技能的人。

5) 开发环境风险

(1) 设施未及时到位；

(2) 设施虽到位，但不配套，如没有电话、网线、办公用品等；

(3) 设施拥挤、杂乱或者破损；

(4) 开发工具未及时到位；

(5) 开发工具不如期望的那样有效，开发人员需要时间创建工作环境或者切换新的工具；

(6) 新的开发工具的学习期比预期的长，内容繁多。

6) 客户风险

(1) 客户对于最后交付的产品不满意，要求重新设计和重做；

(2) 客户的意见未被采纳，造成产品最终无法满足用户要求必须重做；

(3) 客户对规划、原型和规格的审核决策周期比预期的要长；

(4) 客户没有或不能参与规划、原型和规格阶段的审核，导致需求不稳定和产品生产周期的变更；

(5) 客户答复的时间(如回答或澄清与需求相关问题的时间)比预期长；

(6) 客户提供的组件质量欠佳，导致额外的测试、设计和集成工作，以及额外的客户关系管理工作。

7) 产品风险

(1) 矫正质量低下的不可接受的产品，需要比预期更多的测试、设计和实现工作；

(2) 开发额外的不需要的功能，延长了计划进度；

(3) 严格要求与现有系统兼容，需要进行比预期更多的测试、设计和实现工作；

(4) 要求与其它系统或不受本项目组控制的系统相连，导致无法预料的设计、实现和测试工作；

(5) 在不熟悉或未经检验的软件和硬件环境中运行所产生的未预料到的问题；

(6) 开发一种全新的模块将比预期花费更长的时间；

(7) 依赖正在开发中的技术将延长计划进度。

8) 设计和实现风险

(1) 设计质量低下，导致重复设计；

(2) 一些必要的功能无法使用现有的代码和库实现，开发人员必须使用新的库或者自行开发新的功能；

(3) 代码和库质量低下，导致需要进行额外的测试，修正错误或重新制作；

(4) 过高估计了增强型工具对计划进度的节省量；

(5) 分别开发的模块无法有效集成，需要重新设计或制作。

9) 过程风险

(1) 大量的纸面工作导致进程比预期的慢；

(2) 前期的质量保证行为不真实，导致后期的重复工作；

(3) 太不正规(缺乏对软件开发策略和标准的遵循)，导致沟通不足，质量欠佳，甚至需要重新开发；

(4) 过于正规(教条地坚持软件开发策略和标准)，导致过多耗时于无用的工作；

(5) 向管理层撰写进程报告占用开发人员的时间比预期的多；

(6) 风险管理粗心，导致未能发现重大的项目风险。

10.5.4　软件风险管理模型

1. Boehm 模型

Boehm 用公式 RE=P(UO)*L(UO)对风险进行定义，其中 RE 表示风险或者风险所造成的

影响，P(UO)表示令人不满意的结果所发生的概率，L(UO)表示糟糕的结果会产生的破坏性的程度。在风险管理步骤上，Boehm 基本沿袭了传统的项目风险管理理论，指出风险管理由风险评估和风险控制两大部分组成，风险评估又可分为识别、分析、设置优先级 3 个子步骤，风险控制则包括制定管理计划、解决和监督风险 3 步。

Boehm 思想的核心是 10 大风险因素列表，其中包括人员短缺、不合理的进度安排和预算、不断的需求变动等。针对每个风险因素，Boehm 都给出了一系列的风险管理策略。在实际操作时，以 10 大风险列表为依据，总结当前项目具体的风险因素，评估后进行计划和实施，在下一次定期召开的会议上再对这 10 大风险因素的解决情况进行总结，产生新的 10 大风险因素表，依此类推。

10 大风险列表的思想可以将管理层的注意力有效地集中在高风险、高权重、严重影响项目成功的关键因素上，而不需要考虑众多的低优先级的细节问题。而且，这个列表是通过对美国几个大型航空或国防系统软件项目的深入调查，编辑整理而成的，因此有一定的普遍性和实际性。

2．CRM 模型

SEI 提出了持续风险管理模型(Continuous Risk Management，CRM)。

SEI 的风险管理原则是：不断地评估可能造成恶劣后果的因素；决定最迫切需要处理的风险；实现控制风险的策略；评测并确保风险策略实施的有效性。

CRM 模型要求在项目生命期的所有阶段都关注风险识别和管理。它将风险管理划分为 5 个步骤：风险识别、分析、计划、跟踪、控制。图 10.5 所示的框架显示了应用 CRM 的基础活动及其之间的交互关系，强调了这是一个在项目开发过程中反复持续进行的活动序列。每个风险因素开展的不同活动可以是并发的或者交替的。

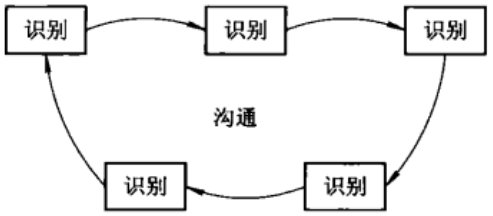


图 10.5 SET 风险管理范例

图 10.5 中的箭头标识了信息的逻辑流，而沟通则是信息流的核心和手段。其中，风险识别依靠问卷完成，问卷覆盖了大概 200 个问题，共涉及 13 个主要领域。风险分析侧重于理解每个风险在该项目中的发生几率和后果严重性，从而产生最严重的 10 大风险问题。风险计划是将如下内容文档化：风险管理步骤的描述、负责人及其职责、行为执行和完结的时间，并且确定风险处理的优先级，制定整体的管理计划。风险跟踪是获取、整理并汇报 10 大风险问题当前的状态，其目的是收集精确的、及时的和相关的信息，并将它们表达成容易理解的方式提交给负责人。风险控制是为了根据风险及其缓解计划进行及时而有效的决策，具体操作包括分析风险跟踪阶段产生的风险状态信息，明确地决定采取什么行动并实现它们。而处于核心地位的沟通则强调其有效性和针对性，要注意将合适的信息传达给合适的组织层次以得到最有效的分析和管理，这些层次包括开发方和用户方双方的组织结构。

3. Leavitt 模型

SEI 和 Boehm 模型都以风险管理的过程为主体，研究每个步骤所需的参考信息及其操作。而 Aalborg 大学提出的思路则是以 Leavitt 模型为基础，着重从导致软件开发风险的不同角度出发探讨风险管理。

1964 年提出的 Leavitt 模型将形成各种系统的组织划分为 4 个有趣的组成部分：任务、结构、角色和技术。这 4 个组成部分和软件开发的各因素很好地对应起来：角色覆盖了所有的项目参与者，例如软件用户、项目经理和设计人员等；结构表示项目组织和其它制度上的安排；技术则包括开发工具、方法、硬件软件平台；任务描述了项目的目标和预期结果。Leavitt 模型的关键思路是：模型的各个组成部分是密切相关的，一个组成部分的变化会影响其它的组成部分。如果一个组成部分的状态和其它的状态不一致，就会造成比较严重的后果，并可能降低整个系统的性能。

10.6 思考与习题

1. 简述软件测试管理过程及其主要功能。
2. 软件风险分析的目的是什么？
3. IEEE 软件测试计划文档模板规定了哪些测试方面的相关内容？
4. 测试管理的主要内容是什么？
5. 需求管理是什么？
6. 配置管理具有哪些关键活动？
7. 如何认识软件缺陷管理？

第 11 章 软件测试工具

本章介绍了软件测试工具。对于当前软件测试工具的类型、软件测试工具具有的共有特征以及如何选择软件测试工具等几个方面进行了说明。

11.1 软件测试工具概述

通过软件测试工具能够使软件的一些简单问题直观地显示在读者的面前，这样能使测试人员更好地找出软件错误的所在。其存在的价值是为了提高测试效率，用软件来代替一些人工输入。

软件测试工具可实现人工无法实现的测试功能，发现人工测试中很难发现的缺陷，减少测试执行时间，提高测试效率。软件测试工具也有如下不足：

(1) 某些测试工具难于学习和使用，创建和修改测试脚本费时费力，相对人工测试而言，不一定节省时间。

(2) 测试工具只能解决某一方面的问题，应用范围狭窄，应根据测试实际需要确定是否选用和选用什么样的测试工具。

(3) 某些商业测试工具售价高昂。

11.2 软件测试工具分类

根据分类方法标准不同，软件测试工具有如下多种分类。

11.2.1 按测试工具所属公司分类

下面介绍几大公司的软件测试产品。

1. MI 公司产品

MI 公司的全称是 Mercury Interactive，作为软件测试工具领域的第一大公司，在市场上占绝对的主导地位。MI 的 4 大产品 LoadRunner、QTP、WinRunner 和 TestDirector 在全球市场的占有率高达 55%。

1) LoadRunner

LoadRunner 属于性能测试工具，用于 C/S 和 B/S 模式的 Web 系统测试。它通过模拟虚拟并发用户数实施压力测试，预测系统行为和性能的负载，对整个软件架构测试分析。LoadRunner 可运行在 Windows、Linux 等多种操作系统上，目前流行的版本是 LoadRunner 8.0。

2) QTP

QTP(QuickTest Professional)是一种自动测试工具，使用 QTP 的目的是用它来执行重复

的手动测试，主要用于回归测试和测试同一软件的新版本。在测试前要考虑好如何对应用程序进行测试，例如要测试哪些功能、操作步骤、输入数据和期望的输出数据等。

QTP 9.0 是一个 B/S 系统的自动化功能测试的利器。QTP 9.0 覆盖了当前绝大多数的软件开发技术，简单高效，并具备测试用例可重用的特点。

3) WinRunner

WinRunner 是一种企业级的功能测试工具，用于检测应用程序是否能够达到预期的功能及正常运行。通过自动录制、检测和回放用户的应用操作，WinRunner 能够有效地帮助测试人员对复杂的企业级应用的不同发布版进行测试，提高测试人员的工作效率和质量，确保跨平台的、复杂的企业级应用无故障发布及长期稳定运行。

WinRunner 具有如下功能：

(1) 轻松创建测试。用 WinRuuner 创建一个测试，只需点击鼠标和键盘，完成一个标准的业务操作流程，WinRunner 会自动记录你的操作并生成所需的脚本代码。这样，即使计算机技术有限的业务用户也可轻松创建完整的测试。用户还可以直接修改测试脚本以满足各种复杂测试的需求。WinRunner 提供了两种测试创建方式，以满足测试团队中业务用户和专业技术人员的不同需求。

(2) 插入检查点。在记录一个测试的过程中，可以插入检查点，检查在某个时刻/状态下，应用程序是否运行正常。在插入检查点后，WinRunner 会收集一套数据指标，在测试运行时对其一一验证。WinRunner 提供了几种不同类型的检查点，包括文本、GUI、位图和数据库。例如，使用位图检查点，你可以检查公司的图标是否出现于指定位置。

(3) 检验数据。除了创建并运行测试，WinRunner 还能验证数据库的数值，从而确保业务交易的准确性。例如，在创建测试时，设定哪些数据库表和记录需要检测；在测试运行时，测试程序会自动核对数据库内的实际数值和预期的数值。WinRunner 自动显示检测结果，在有更新/删除/插入的记录上突出显示以引起注意。

(4) 增强测试。为了彻底全面地测试一个应用程序，需要使用不同类型的数据来测试。WinRunner 的数据驱动向导使用户只需简单地点击几下鼠标，就可以把一个业务流程测试转化为数据驱动测试，从而反映多个用户各自独特且真实的行为。

(5) 运行测试。创建好测试脚本，并插入检查点和必要的添加功能后，即可执行测试。WinRunner 会自动操作应用程序，根据业务流程执行每一步的操作。

(6) 分析结果。测试运行结束后，需要分析测试结果。WinRunner 通过交互式的报告工具来提供详尽的、易读的报告。报告中会列出测试中发现的错误内容、位置、检查点和其它重要事件，帮助分析测试结果。

(7) 维护测试。随着时间的推移，开发人员会对应用程序做进一步的修改，并需要增加另外的测试。WinRunner 不必对程序的每一次改动都重新创建测试，可以创建在整个应用程序生命周期内都能重复使用的测试，以节省时间和资源。每次记录测试时，WinRunner 会自动创建一个 GUI Map 文件以保存应用对象。这些对象分层次组织，既可以总览所有的对象，也可以查询某个对象的详细信息。

4) TestDirector

TestDirector 是基于 Web 集成的测试管理工具，用于组织和管理整个测试过程。作为业界第一个 Web 形式的测试管理系统，TestDirector 可以在公司组织内进行全球范围内测试的

协调。通过在一个整体的应用系统中集成了测试需求管理、测试计划、测试日程控制以及测试执行和错误跟踪等功能，TestDirector 可极大地加速测试过程。

TestDirector 消除了组织机构间、地域间的障碍，使得测试人员、开发人员或其他 IT 人员通过中央数据仓库在不同位置能互通测试信息。TestDirector 将测试过程进行流水作业：从测试需求管理到测试计划、测试日程安排、测试执行以至出错后跟踪，仅在一个基于浏览器的应用中便可完成。

(1) 需求管理：通过提供一个直观机制将需求和测试用例、测试结果和报告错误联系起来，从而确保完全的测试覆盖率。

(2) 计划测试：Test Plan Manager 指导测试人员如何将应用需求转化为具体的测试计划，帮助定义测试应用程序，明确任务和责任。

(3) 安排和执行测试：测试计划建立好后，TestDirector 的测试实验室管理为测试日程制订提供一个基于 Web 的框架，根据测试计划中创立的指标对测试的执行进行监控。

(4) 出错管理：TestDirector 将出错管理直接作用于测试的全过程，提供管理系统终端与终端的出错跟踪，帮助发现问题、修改错误，直到检验修改结果。

(5) 图形化和报表输出：图表和报告在测试的任一环节对数据信息进行分析。

2. IBM Rational 公司产品

从项目设计到实现，IBM Rational 软件交付平台为软件和基于软件系统的开发提供了完整解决方案的集成平台。IBM Rational 系列产品如表 11.1 所示。

表 11.1 IBM Rational 系列产品介绍

系列	产品名称	产品说明
需求分析	IBM Rational RequisitePro	主要关注于项目的文档、通信和控制的不断变化的需求
	IBM Rational Software Modeler	使用统一建模语言(UML)符号可视化模型系统和应用程序
设计和构建	IBM Rational Software Architect	使用一个工具统一架构、设计和开发
	IBM Rational Application Developer for WebSphere Software	
	IBM Rational Web Developer for WebSphere Software	利用一个易于使用的 IDE 构建、测试和部署 Web、Web 服务和 Java 应用程序，这个 IDE 是为了使用 IBM WebSphere 软件而优化的，而且它还提供了在其它技术平台上进行开发的能力
	IBM Rational Rose	IBM Rational Rose 使构架设计师和设计人员能够使用统一建模语言(UML)进行模型驱动开发。这些用户可以建立软件构架、业务需求、可重用资源、管理级通信的平台独立模型。行业标准的 UML 支持和强大的模式引擎可以创建语义丰富的应用程序构架，该构架可满足业务需求，易于为开发团队理解

续表

系列	产 品 名 称	产 品 说 明
软件质量保证	IBM Rational Functional Tester	执行功能测试自动化，以增加复杂 Java、Microsoft Visual Studio.NET WinForm 和基于 Web 的应用程序中发现的缺陷
	IBM Rational Performance Tester	基于多用户负载，验证 Web 应用程序的性能、可伸缩性和可靠性
	IBM Rational PurifyPlus	自动化运行时错误侦测、瓶颈问题发现和代码覆盖分析，以提高单位测试和调试的有效性
	IBM Rational Test RealTime	执行面向内嵌的和其它实时环境的应用程序的组件测试和运行分析
软件配置管理	IBM Rational ClearCase	全面的版本控制、工作空间管理以及构建和版本管理
	IBM Rational ClearQuest	通过开发生命周期来管理工作流和变更请求
过程和项目管理	IBM Rational Unified Process	采用灵活的、可配置的开发过程
	IBM Rational Method Composer	采用灵活的、可配置的开发过程
	IBM Rational Portfolio Manager	自动化 IT 管理过程并调整优先级、项目和带有商业目标的人
	IBM Rational SUMMIT Ascendant	使用方法库和基于 Web 的用于规划、评估和监控的工具，来规划和管理企业 IT 项目和规划
	IBM Rational Team Unifying Platform	提供核心基础设施，包括生命周期过程指南、需求管理、版本控制、缺陷和变更跟踪、测试管理以及项目度量和报告

IBM Rational 公司的测试工具主要有以下 4 款：

- Rational Testmanager(测试管理工具)
- Rational Robot(功能/性能工具)
- Rational Purify(白盒测试工具)
- Rational ClearQuest(缺陷管理工具)

在实际项目管理中，ClearQuest 主要用作记录 3 种活动：BaseCMActivity、Enhancement 和 Defect。BaseCMActivity 表示基本配置管理活动，Enhancement 用于涉众请求或新需求，Defect 表示缺陷。具体如下所示。

1) 指派任务

通过 ClearQuest 生成项目的任务工单，将工单指派给项目小组成员(ClearQuest 与 MS-Project 集成，可直接将计划进度表导入 ClearQuest 生成任务工单)。项目组成员可以在 ClearQuest 设置查询条件，显示工单(ClearQuest 与 ClearCase 集成，项目组成员可在各自的视图上查找本人的任务工单)。项目组成员激活工单，并根据工单内容进行工作和提交工作产出。ClearQuest 自动对项目组成员的工作进行跟踪，记录任务的开始时间、结束时间、工单状态、工作产出等。

通过 ClearQuest 指派任务和记录，项目经理较为方便地了解各项任务的实现情况和项目的状态，统计和分析项目数据及获取项目经验数据。

2) 获取需求

开发人员在设计、编码过程中，发现用户提出的需求不合理或存在可改进之处，都可以通过 ClearQuest 记录并提交请求。

选择 ClearQuest 的 EnhancementRequest，可以输入涉众请求(新需求或变更需求)，并在表单中填写需求描述、重要性、客户信息等相关信息。涉众请求提交之后，项目经理或需求分析人员可以定期召集项目组成员讨论和审核，一旦确定下来，需求管理人员可将涉众请求汇总成需求文档，并加入 RequisitePro 进行管理。

3) 缺陷跟踪

利用 ClearQuest 可以管理项目开发和测试中发现的缺陷。测试人员通过 ClearQuest 提交测试过程中发现的缺陷记录，或者质量保证人员提交在项目开发过程中发现的任何问题，提交时必须详细填写缺陷的相关内容，包括缺陷的描述、属性、修复时间、修复人员、状态和测试记录等。项目经理确认无误后再分配任务，将此缺陷记录指派给项目组成员，项目组成员根据缺陷的具体描述进行修复，并以此修改缺陷的状态。

利用 ClearQuest，项目管理人员可以随时查看缺陷的历史记录和缺陷修复状况，并生成报告。

4) 管理需求

使用 ClearQuest 可以加强项目需求的收集，包括用户提出的新需求，还有开发人员在设计、编码过程中发现的不合理或可优化的方面，都可以通过 ClearQuest 记录并提交请求。待讨论和审核之后，可以汇总成需求文档，加入 RequisitePro 进行管理。

ClearQuest 与 MS-Project、ClearCase 集成，功能更加强大，并且 ClearQuest 公开了部分的接口和源代码，用户可以修改 ClearQuest 记录和显示的内容，以及显示界面和风格。

3. Compuware 公司产品

Compuware 公司为全球计算机用户的应用系统提供从开发、继承、测试、运行、管理到维护的全方位保障和服务。其开发的测试工具主要有：

- QACenter(测试管理测试工具)
- TrackRecord(缺陷管理测试工具)
- QARun(功能测试工具)
- QALoad(性能测试工具)

4. 开放源代码的软件测试工具

下面介绍一些运用较为广泛的开放源代码的测试工具，如 Junit、Splint 和 JMeter 等。

1) 单元测试工具 Junit

Junit 是一个开源的 Java 编程语言的单元测试框架，由 Erich Gamma 和 Kent Beck 发明。作为一个应用程序的半成品，Junit 提供了应用程序之间可共享的结构。开发者只需把 Junit 框架融入到应用程序中并加以扩展，以满足特定需要。Junit 框架和其它软件测试工具包的不同之处在于，框架提供了一致的结构，而不仅仅是一组工具类。本书 15.1 节对 Junit 作全面详细介绍。

2) 性能测试工具 JMeter

Apache JMeter 用于测试软件的功能特性，度量被测试软件的性能，针对静态资源和动态资源(如 Servlets、Perl 脚本、Java 对象、数据查询、FTP 服务等)的性能测试。Jmeter 通过模拟大量的服务器负载、网络负载、软件对象负载，全面测试软件的性能。

参考网址：http://jakarta.apache.org/jmeter/。

3) 缺陷管理工具 Mantis

Mantis 是一款基于 Web 的软件缺陷管理工具，配置和使用都很简单，适合中小型软件开发团队。

使用环境：MySQL，PHP。

参考网址：http://mantisbt.sourceforge.net/。

4) 测试管理工具 TestLink

TestLink 是基于 Web 的测试管理和执行系统。测试小组在系统中可以创建、管理、执行、跟踪测试用例，并且提供在测试计划中安排测试用例的方法。

使用环境：Apache，MySQL，PHP。

参考网址：http://testlink.sourceforge.net/docs/testLink.php。

11.2.2　按测试工具的功能分类

根据测试工具的不同功能进行如下分类：

(1) 测试过程生成器：需求管理工具与基于需求说明书的测试过程生成器联成一体，当需求管理工具捕捉到需求信息后，这些信息会被测试过程生成器利用，生成器通过统计、计算或者探索式的方法创建测试过程。若使用统计的方法生成测试过程，工具会按一个分布选择输入值，这个分布可能是统计上的随机分布，或者是和正在测试的软件的用户配置相匹配的分布。测试数据生成器最常使用的策略是动作、数据、逻辑、事件和状态驱动，因此常用于检测不同种类的软件缺陷。

(2) 代码(测试)覆盖率分析器和代码测量器：此类工具能够量化设计的复杂度，限制测试所必需的集成测试的数量，有助于进行集成测试。此外，工具还能用多种方式(包括代码段、分支段和条件值覆盖率)测量测试覆盖率，有助于把没有覆盖到的分支和逻辑结构加入到测试集中。

(3) 内存泄露检测工具：此类工具用于验证应用程序是否正确地使用它的内存资源，确定应用程序是否释放了它所申请的内存，并且提供运行时的错误检测。因为许多程序缺陷都和内存问题有关，其中包括性能问题，所以如果应用程序对内存的操作非常频繁，进行内存检测是非常必要的。

(4) 度量报告工具：此类工具读取源代码并显示度量信息，如数据流、数据结构和控制流的复杂度，根据模块、操作数、操作符和代码的数量提供代码规模的度量分析数据。

(5) 使用性测试工具：用于测试软件系统界面的易用性以及其它一些特征。

(6) 测试数据生成器：此类工具通过自动生成测试数据来辅助测试过程。目前，市场上有多种工具支持生成测试数据。无论测试数据是用于功能测试、数据驱动的负载测试，还是性能测试和强度测试，测试数据生成器都能够根据一组规则快速地生成测试数据库。

(7) 测试管理工具：此类工具支持对测试生命周期的所有方面进行计划、管理和分析。

(8) 网络测试工具：用于对网络的性能进行监控、测试和诊断。

(9) GUI 测试工具：此类工具通常使用“记录和回放”功能，测试人员在不同环境下创建(记录)、修改和运行(回访)自动化的测试。

(10) 负载/性能和强度测试工具：负载/性能测试工具模拟系统的真实负载，检查系统或者应用程序的响应时间和负载能力。强度测试工具模拟高强度场景运行来确定软件是否会崩溃和什么时候崩溃。

(11) 专用工具：针对特殊的架构或技术进行专门测试的工具，需要对构架上的特殊部分进行专门测试。例如，针对 Web 应用程序是否存在空链接等进行测试。

11.2.3 按测试工具在软件测试中应用的阶段分类

根据测试工具在软件测试中应用的阶段分类，软件测试工具分为黑盒测试工具、白盒测试工具和测试管理工具 3 类。

1. 黑盒测试工具

黑盒测试工具是指测试软件功能或性能的工具，主要用于系统测试和验收测试。黑盒测试工具可分为功能测试工具和性能测试工具。

2. 白盒测试工具

白盒测试工具应用在具有高可靠性的软件领域，例如军工软件、航天航空软件、工业控制软件等。白盒测试工具是对源代码进行测试，主要测试词法分析与语法分析、静态错误分析、动态检测等。目前白盒测试工具主要支持标准 C、C++、Visual C++、Java、Visual J++ 等程序开发语言。

根据测试工具原理不同，白盒测试又分为静态测试工具和动态测试工具。

静态测试工具直接对代码进行分析，不需要运行代码，也不需要对代码编译链接和生成可执行文件。静态测试工具一般是对代码进行语法扫描，找出不符合编码规范的地方，根据某种质量模型评价代码的质量，生成系统的调用关系图等。静态测试工具的代表有 Telelogic 公司的 Logiscope 软件、PR 公司的 PRQA 软件。

动态测试工具一般采用“插桩”的方式，向代码生成的可执行文件中插入一些监测代码，用来统计程序运行时的数据。其与静态测试工具最大的不同就是动态测试工具要求被测系统实际运行。动态测试工具的代表有 Compuware 公司的 DevPartner 软件、Rational 公司的 Purify 系列。

下面介绍一些流行的白盒测试工具。

1) Parasoft 白盒测试工具集

Parasoft 白盒测试工具集包括如下工具：Jtest 用于 Java 的代码分析和动态类、组件测试；Jcontract 用于 Java 的实时性能监控以及分析优化；C++ Test 用于 C 和 C++ 语言的代码分析和动态测试；CodeWizard 用于 C 和 C++ 的代码静态分析；Insure++ 用于 C 和 C++ 的实时性能监控以及分析优化。

2) NuMega DevPartner Studio 白盒测试工具集

NuMega DevPartner Studio 白盒测试工具集主要用于代码开发阶段，检查应用的可靠性

和稳定性。产品具有错误检测、性能分析、代码覆盖分析等功能，适合捕获、定位错误，抽取代码执行频度，以及抽取代码覆盖率等数据。产品如下所示：

(1) BoundsChecker。BoundsChecker 提供清晰的程序错误分析，能自动查明静态的堆栈错误及内存/资源泄露，迅速定位出错源代码。BoundsChecker 错误检测范围主要包括：指针和泄露错误、内存错误、API 和 OLE 错误。

(2) TrueCoverage。TrueCoverage 能够列出所有函数被调用次数、所占比率等，并直接定位到源代码。

(3) TrueTime。TrueTime 用于查找和修改性能瓶颈，自动定位到运行缓慢的代码，调整整个代码性能。

(4) SmartCheck。SmartCheck 作为 Visual Basic 的 runtime 调试工具，检测所有的 Windows API 函数调用、内存分配以及一些重要的程序错误。SmartCheck 检错的种类包括泄露、接口方法失败、存储错误、程序和函数失败及 runtime 错误等。

(5) FailSafe。FailSafe 通过插装的代码捕获，记录执行时程序和系统的重要信息，指出错误发生时程序和系统的状态。

(6) CodeReview。CodeReview 作为 Visual Basic 的自动源代码分析工具，对应用程序的组件、逻辑错误、应用程序性能和可用性问题、Windows API 调用和标准一致性等问题进行源代码检查。

(7) JCheck。JCheck 提供功能强大的图形化线程和事件分析工具，通过生动的图形化方法表现程序的线程状态信息以及和 Windows 线程、同步对象、线程组等的交互作用信息，使开发人员能够直观地分析 Java Applet 或 Application，确定 runtime 错误，对执行和逻辑错误进行分析，发现死锁、活锁、资源缺乏和系统失败，诊断线程同步等问题。

3) 其它公司的白盒测试工具集

- Compuware 白盒测试工具集
- BoundsChecker C++，Delphi API 和 OLE 错误检查、指针和泄露错误检查、内存错误检查
- TrueTime C++，Java，Visual Basic 代码运行效率检查、组件性能的分析
- FailSafe Visual Basic 自动错误处理和恢复系统
- Jcheck M$ Visual J++ 图形化的纯种和事件分析工具
- TrueCoverage C++，Java，Visual Basic 函数调用次数、所占比率统计以及稳定性跟踪
- CodeReview Visual Basic 自动源代码分析工具
- Xunit 白盒测试工具集

3. 测试管理工具

测试管理工具是指管理整个测试流程的工具，主要功能有测试计划的管理、测试用例的管理、缺陷跟踪、测试报告管理等，一般贯穿于整个软件测试生命周期。

总之，测试工具实现原理基本相同，没有本质区别。学习测试工具，应先以主流的测试工具为模板，如性能测试工具就选 LoadRunner，功能测试工具就选 QTP，测试管理就选 TestDirector 等，其它的测试工具就可触类旁通。

11.3 软件测试工具特征

软件测试工具具有如下特征：

1. 支持脚本语言、函数库

支持脚本语言和函数库是测试工具最基本的要求。如果程序作了修改，只需把原脚本中的相应函数进行更改，而不用改动所有可能的脚本，这样可节省大量工作。另外，通过对外部函数的支持，如对 DLL 文件的访问，对数据库编程接口的调用，可获得强大的功能。

2. 对程序界面中对象的识别能力

测试工具必须能够将程序界面中的相应对象(如按钮、文本框、表单等)区分并识别，录制的测试脚本才能具有良好的可读性、修改的灵活性和维护的方便性。如果只是简单通过像素位置坐标区分对象，则会存在较多问题，例如界面稍微改变，或者屏幕的分辨率、测试环境的改变，会导致原有的测试脚本无法使用。

3. 抽象层

抽象层和对象识别能力有一定的关系。在捕捉回放程序界面过程中，抽象层一般位于被测应用程序和录制生成的测试脚本之间，抽象层用于将程序界面中存在对象实体映射成逻辑对象，测试针对逻辑对象进行，不需依赖界面的对象实体，从而减少测试脚本建立和维护的工作量。

4. 分布式测试的网络支持

互联网软件，如网络会议系统、远程培训系统、聊天系统等，一般具有协同工作、相互通信等模式，支待多用户共同操作。对这类软件进行测试时有如下要求：

(1) 测试工具在进行测试时传输的数据量要小，具有独立性，避免被测软件影响。

(2) 按照事先设置的任务执行时间表进行，即在指定时间、指定设备上执行指定的测试任务。

(3) 当两个测试任务并发时，需要能保持协调或协同处理，避免出现资源竞争问题。

5. 图表功能

测试工具可将测试结果生成相关统计报表，利于测试人员的工作。

6. 测试工具的集成能力

测试工具的引入是一个长期的过程，是伴随着测试过程改进的一个持续的过程。因此，测试工具应与开发工具进行良好的集成，并且能够和其它测试工具集成。

11.4 软件测试工具选择

当前市场上测试工具很多，每个测试工具在不同环境有其各自的优点和缺点。如何选择最佳的测试工具，主要依赖于系统工程环境以及组织特定的其它需求和标准。因此，选择自动化测试工具应从以下几个方面考虑：

(1) 确定测试生命周期工具类型。确定测试工具与操作系统、编程语言环境和其它方面

相兼容。

(2) 确定各种系统构架。必须确定应用程序在技术上的构架，其中包括整个组织或者项目使用的中间件、数据库、操作系统、开发语言、使用的第三方插件等。

(3) 确定被测试应用程序管理数据的方式。必须了解被测试应用程序管理数据的方式，确定自动测试工具如何支持对数据的验证。

(4) 确定测试类型。必须了解工具的测试类型，如用于回归测试、强度测试或者容量测试等测试工具功能差距较大。

(5) 确定项目进度。测试工具是否影响测试进度，在进度时间表内，评审测试人员是否有足够的时间学习使用测试工具非常重要。

(6) 确定项目预算。

第 12 章 测试管理工具

测试管理工具指利用工具对测试输入、执行和结果进行管理。一般而言，测试管理工具包括对测试计划、测试用例、测试实施进行管理。

本章主要介绍 TestDirector 测试管理工具。

12.1 测试管理工具概述

当前市场上的各种测试管理工具，如 TestManager、Wiki、Bugzilla+Test Runner、TestDirector、TestLink 以及 word 等，具有各自的优缺点，如表 12.1 所示。

表 12.1 各种测试管理工具

工具名	综述	优 点	缺 点
TestManager	Rational 测试解决方案中推荐的测试管理工具	1. 功能强大 2. 对测试用例无限分级 3. 可以和 Rational 的测试工具 robot、functional 相结合 4. 有测试用例执行的功能，但必须先生成对应的手工或自动化脚本	1. 本地化支持不好，汉字显示太小 2. 测试用例不太稳定 3. 必须安装客户端才可使用，和开发人员交流不方便 4. 测试用例的展示形式单一
Wiki	使用 Wiki 作测试管理工具	1. Web 界面形式，交流方便 2. 测试用例形式多样 3. Wiki 提供测试用例的版本控制、版本比较功能 4. Wiki 提供测试用例加注释功能，方便测试用例评审 5. Wiki 本身强大的全文索引功能 6. 可以任意为测试用例添加标签	1. 并非专业的测试管理工具 2. 无法和其它测试工具集成 3. 测试用例的统计不方便 4. 没有测试用例的执行跟踪功能 5. 有一些 Wiki 本身的限制，如不同产品的测试用例名不能重复 6. 目前还没有定制统一的模板
Bugzilla+Test Runner	开源的测试管理解决方案，有很多开源软件使用此方式管理	1. 开源免费 2. Web 方式的管理界面 3. 自动邮件提醒 4. 与 Bugzilla 结合紧密 5. 测试用例可以分优先级 6. 测试用例可以有评审的功能	1. 安装设置较繁琐 2. 编写测试用例必须按照一个步骤对应一个验证点的形式来编写

续表

工具名	综述	优　点	缺　点
TestDirector		1. 和 Rational 测试工具结合 2. Web 方式的界面 3. 有测试用例执行跟踪的功能 4. 有灵活的缺陷定制 5. 和自身的缺陷管理工具紧密集成 6. 界面较友好	1. 每个项目库同时在线人数有限 2. 存在不稳定性
新版 CQ(7.0)	新版本的 CQ 中增加了测试用例管理的功能	1. 和 CQ 的缺陷管理紧密结合 2. 使用 CQ 强大的查询和图表功能	Eclipse 的界面，较为笨重，需要安装
TestLink		1. Web 方式的界面 2. 可以和 Bugzilla 缺陷管理工具整合 3. 和其它缺陷管理工具整合 4. 同时具有需求管理的功能	
Excel 形式	适合小型项目	依托 Excel 本身的强大功能，很灵活，易于扩展	维护较麻烦，统计、度量等也不方便
Word 形式		依托 Word 本身的强大功能，很灵活，易于扩展	不如 Excel 格式统一，也不如 Excel 容易统计

12.2　测试管理工具——TestDirector

12.2.1　TestDirector 简介

TestDirector 是 HP 公司推出的基于 Web 的测试管理工具，无论是通过 Internet 还是通过 Intranet 都可以以基于 Web 的方式来访问 TestDirector。

TestDirector 能够系统地控制整个测试过程，并创建整个测试工作流的框架和基础，使整个测试管理过程变得更为简单和有组织。

TestDirector 能够维护一个测试工程数据库，并且能够覆盖应用程序功能性的各个方面。工程中的每一个测试点都对应着一个指定的测试需求。TestDirector 提供了直观和有效的方式来计划和执行测试集、收集测试结果并分析数据，还提供了一个完善的缺陷跟踪系统，能够跟踪缺陷从产生到最终解决的全过程。

TestDirector 提供了与许多测试工具，如 WinRunner、LoadRunner 等，以及第三方或者

自主开发的测试工具、需求和配置管理工具、建模工具的整合功能，并提供全套解决方案来进行全部自动化的应用测试。

TestDirector 的测试管理包括如下四个阶段，如图 12.1 所示。

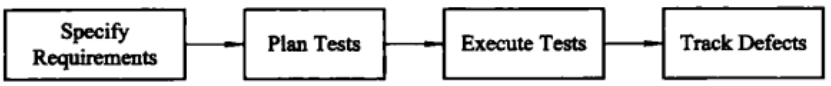


图 12.1 TestDirector 的测试管理流程

- 需求定义(Specify Requirements)：分析应用程序并确定测试需求。
- 测试计划(Plan Tests)： 基于测试需求，建立测试计划。
- 测试执行(Execute Tests)：创建测试集(Test Set)并执行测试。
- 缺陷跟踪(Track Defects)：报告程序中产生的缺陷并跟踪缺陷修复的全过程。

1) 需求定义

该过程用来分析应用程序并确定测试需求，如图 12.2 所示。

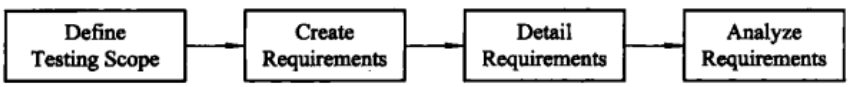


图 12.2 TestDirector 的需求定义

(1) 定义测试范围(Define Testing Scope)：检查应用程序文档，并确定测试范围——测试目的、目标和策略。

(2) 创建需求(Create Requirements)：创建需求树(Requirements Tree)，并确定它涵盖所有的测试需求。

(3) 描述需求(Detail Requirements)：为“需求树”中的每一个需求主题建立了一个详细的目录，并描述每一个需求，给它分配一个优先级，如有必要的话还可以加上附件。

(4) 分析需求(Analyze Requirements)：产生报告和图表来帮助你分析测试需求，并检查需求以确保它们在你的测试范围内。

2) 测试计划

基于已定义的测试需求，创建相应的测试计划，如图 12.3 所示。

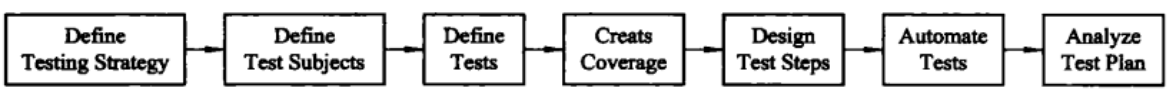


图 12.3 TestDirector 的测试计划

(1) 定义测试策略(Define Testing Strategy)：检查应用程序、系统环境和测试资源，并确认测试目标。

(2) 定义测试主题(Define Test Subject)：将应用程序基于模块和功能进行划分，并对应到各个测试单元或主题，构建测试计划树(Test Plan Tree)。

(3) 定义测试(Define Tests)：定义每个模块的测试类型，并为每一个测试添加基本的说明。

(4) 创建需求覆盖(Create Requirements Coverage)：将每一个测试与测试需求进行连接。

(5) 设计测试步骤(Design Test Steps)：对于每一个测试，先决定其要进行的测试类型(手动测试和自动测试)，若准备进行手动测试，需要为其在测试计划树上添加相应的测试步骤

(Test Steps)。测试步骤描述测试的详细操作、检查点和每个测试的预期结果。

(6) 自动测试(Automate Tests)：对于要进行自动测试的部分，应该利用 MI、自己或第三方的测试工具来创建测试脚本。

(7) 分析测试计划(Analyze Test Plan)：产生报告和图表来帮助分析测试计划数据，并检查所有测试以确保它们满足你的测试目标。

3) 测试执行

创建测试集(Test Set)并执行测试，如图 12.4 所示。

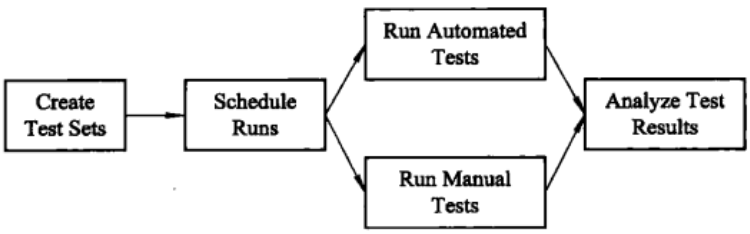


图 12.4　TestDirector 的测试执行

(1) 创建测试集(Create Test Sets)：在你的工程中定义不同的测试组来达到各种不同的测试目标，比如在一个应用程序中测试一个新的应用版本或是一个特殊的功能，并确定每个测试集都包括了哪些测试。

(2) 确定进度表(Schedule Runs)：为测试执行制定时间表，并为测试员分配任务。

(3) 运行测试(Run Tests)：自动或手动执行每一个测试集。

(4) 分析测试结果(Analyze Test Results)：查看测试结果并确保应用程序缺陷已经被发现。生成的报告和图表可以帮助分析这些结果。

4) 缺陷跟踪

报告程序中产生的缺陷并跟踪缺陷修复的全过程，如图 12.5 所示。

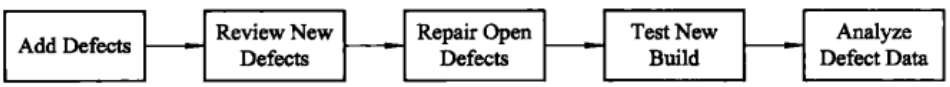


图 12.5　TestDirector 的缺陷跟踪

(1) 添加缺陷(Add Defects)：报告程序测试中发现的新的缺陷。在测试过程中的任何阶段，质量保证人员、开发者、项目经理和最终用户都能添加缺陷。

(2) 检查新缺陷(Review New Defects)：检查新的缺陷并确定哪些缺陷应该被修复。

(3) 修复打开的缺陷(Repair Open Defects)：修复那些你决定要修复的缺陷。

(4) 测试新构建(Test New Build)：测试应用程序的新构建，重复上面的过程，直到缺陷被修复。

(5) 分析缺陷数据(Analyze Defect Data)：产生报告和图表来帮助分析缺陷修复过程，并帮助决定什么时候发布该产品。

12.2.2　TestDirector 的安装

(1) 先安装微软的 IIS 组件。(如已经安装 IIS，可省略此步骤)

(2) 将 TD 安装盘放入光驱，在“我的电脑”中双击光驱，显示其内容，如图 12.6 所示。

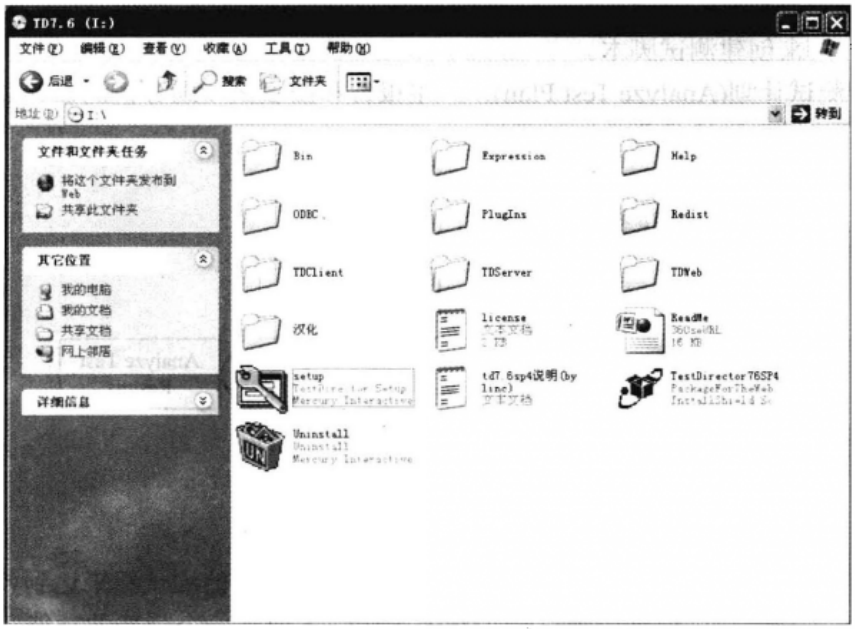


图 12.6 TestDirector 安装图示 1

(3) 双击 setup.exe 文件，运行安装程序，如图 12.7 所示。

图 12.7 TestDirector 安装图示 2

(4) 点击 Next 按钮，如图 12.8 所示。

图 12.8　TestDirector 安装图示 3

(5) 在上面的复选框中打钩，点击 Next 按钮，如图 12.9 所示。

图 12.9　TestDirector 安装图示 4

(6) 选择所要连接的数据库，并点击 Next 按钮，如图 12.10 所示。

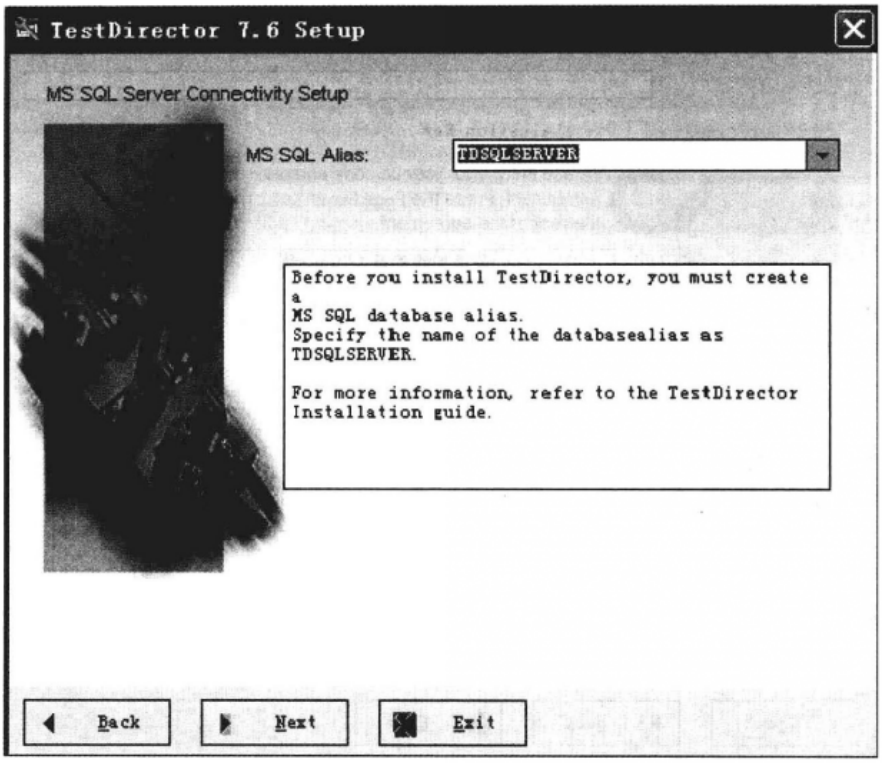


图 12.10 TestDirector 安装图示 5

(7) 创建一个 MS SQL 数据库并点击 Next 按钮，如图 12.11 所示。

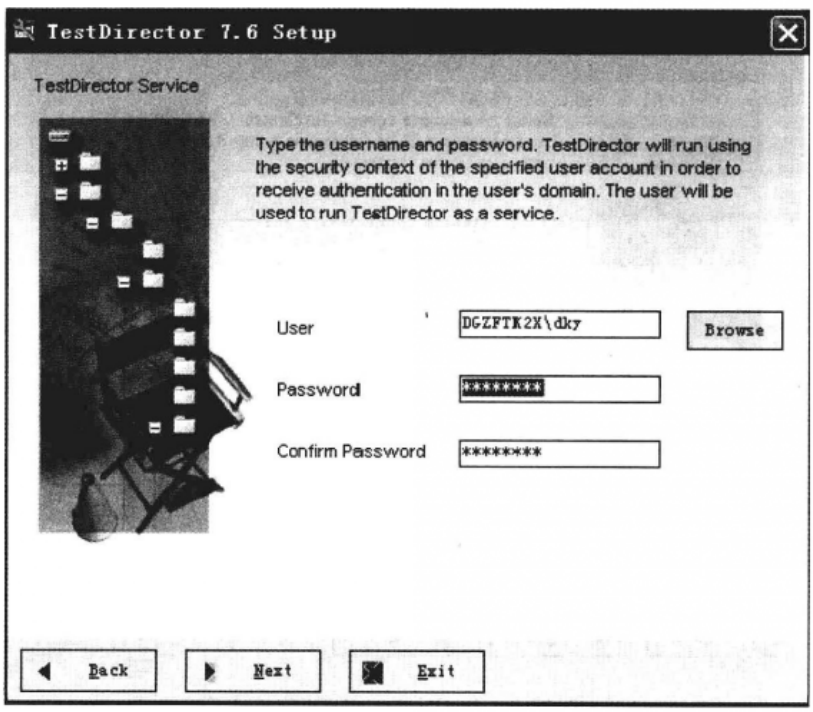


图 12.11 TestDirector 安装图示 6

(8) 在用户中选择一个拥有系统管理员权限的用户，输入正确的密码并确认密码，然后点击 Next 按钮，如图 12.12 所示。

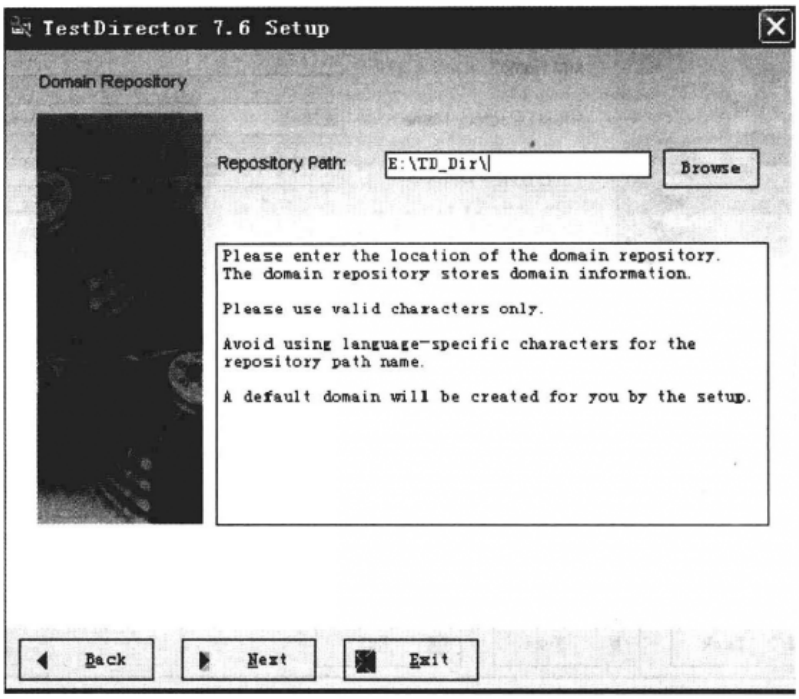


图 12.12　TestDirector 安装图示 7

(9) 输入存储路径或采用 Browse 来选择路径，点击 Next 按钮，如图 12.13 所示。

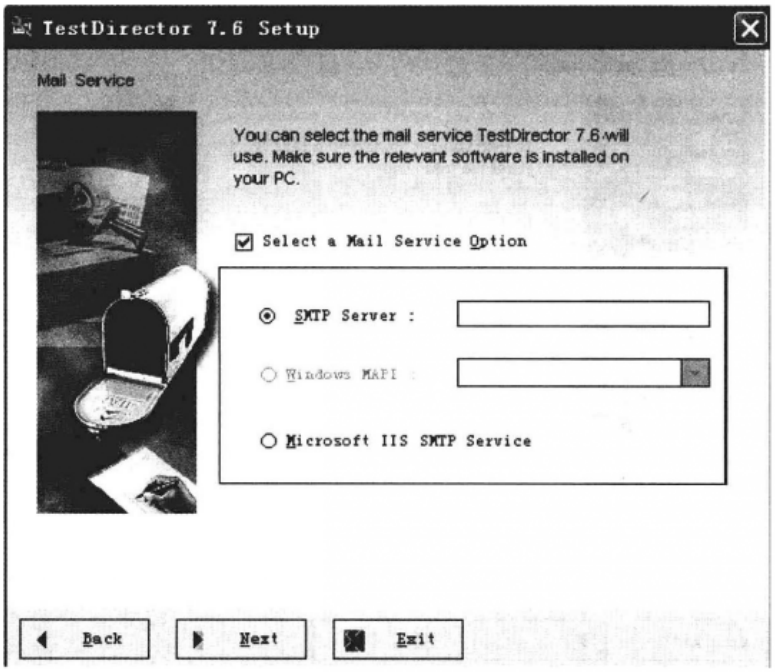


图 12.13　TestDirector 安装图示 8

(10) 选择一种适合自己的 mail 服务，并点击 Next，如图 12.14 所示。

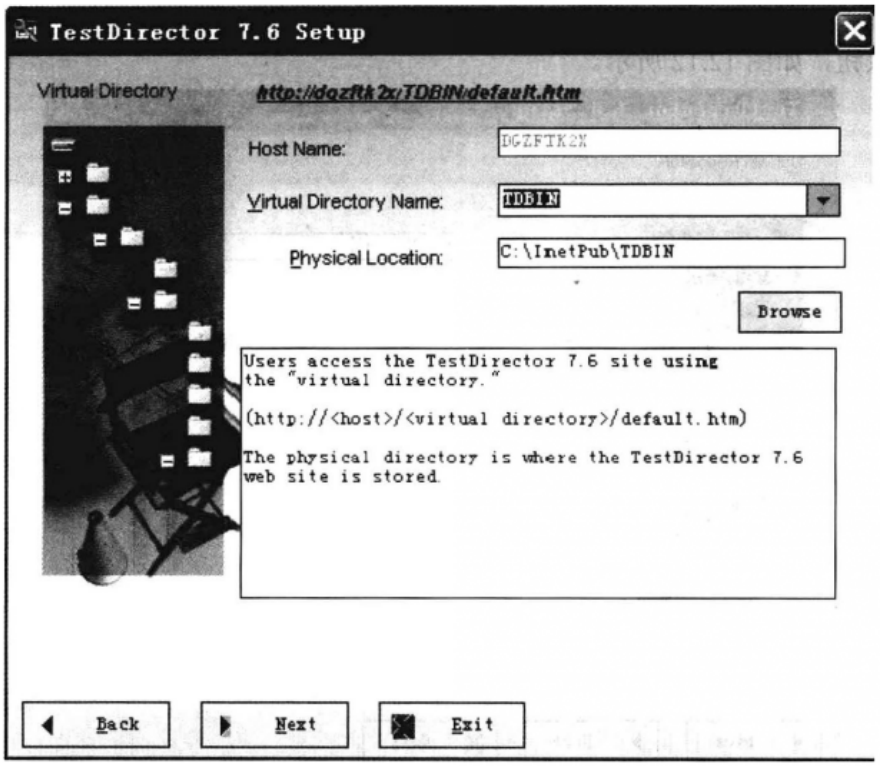


图 12.14 TestDirector 安装图示 9

(11) 选择或输入虚拟路径和物理路径，点击 Next，如图 12.15 所示。

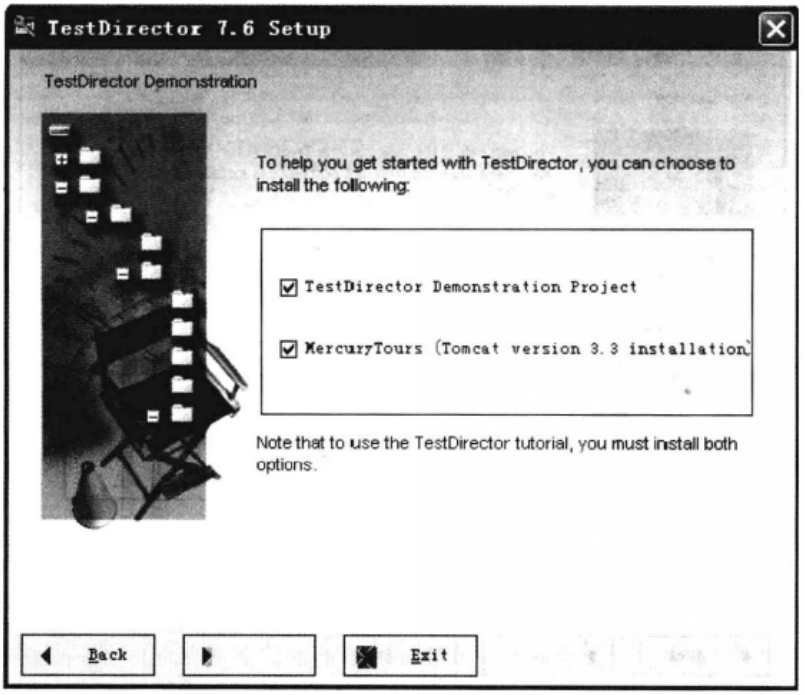


图 12.15 TestDirector 安装图示 10

(12) 保持默认选项点击 Next 按钮，如图 12.16 所示。

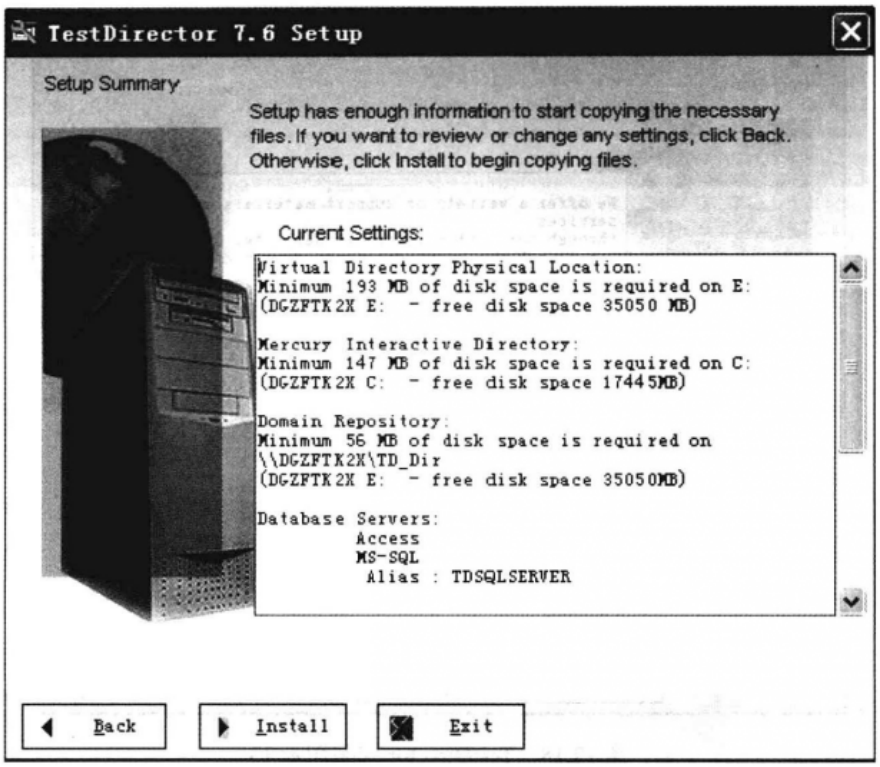


图 12.16　TestDirector 安装图示 11

(13) 安装的基本设置到此完成，点击 Install 按钮进行安装，如图 12.17、12.18 所示。

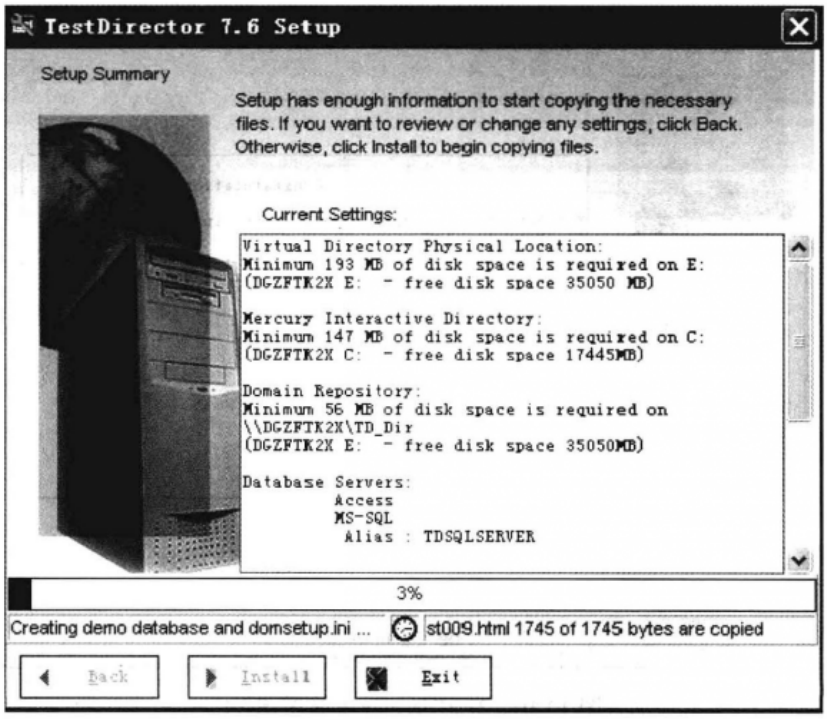


图 12.17　TestDirector 安装图示 12

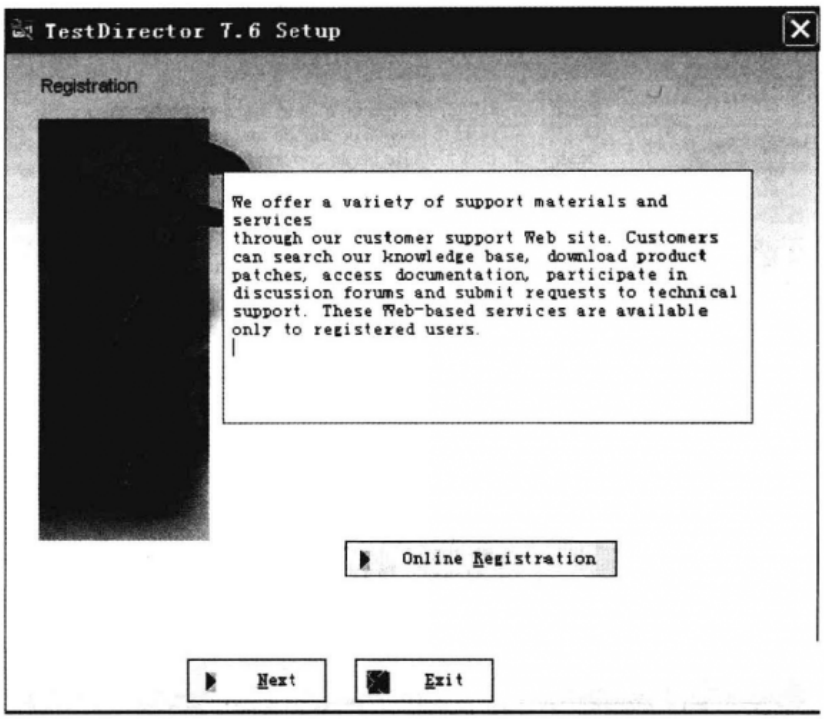


图 12.18 TestDirector 安装图示 13

(14) 安装完成之后点击 Next 按钮，如图 12.19 所示。

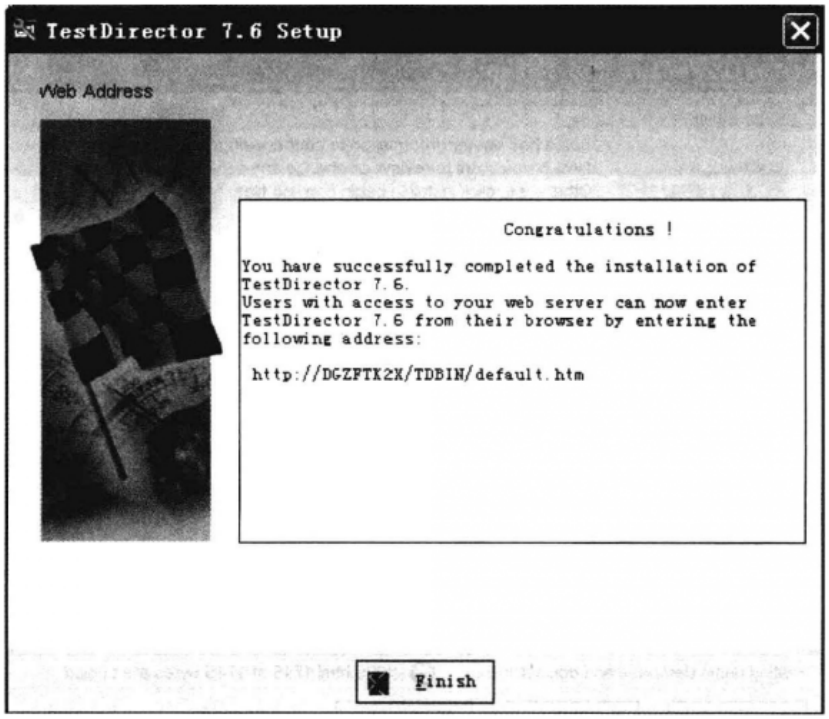


图 12.19 TestDirector 安装图示 14

(15) 安装完成，点击 Finish 按钮，重启后方可生效。

12.2.3　TestDirector 的配置

1．创建项目

(1) 打开浏览器输入 TestDirector URL(http://[server name]/[virtual directory name]/default.htm)访问 TestDirector 选项窗口，如图 12.20 所示。

图 12.20　TestDirector 启动

(2) 点击 Site Administrator 链接打开 Site Administrator 登录窗口，如图 12.21 所示。

图 12.21　打开 Site Administrator 登录窗口

(3) 默认情况下管理员没有定义密码，如果要定义或修改密码，点击 Change Password 链接打开 Site Administrator Password 窗口，可以修改管理员密码，如图 12.22 所示。

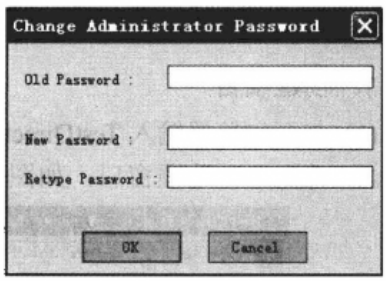


图 12.22 修改密码

(4) 在 Site Administrator 登录窗口中输入密码，点击 Login 按钮打开 Site Administrator 窗口，如图 12.23 所示。

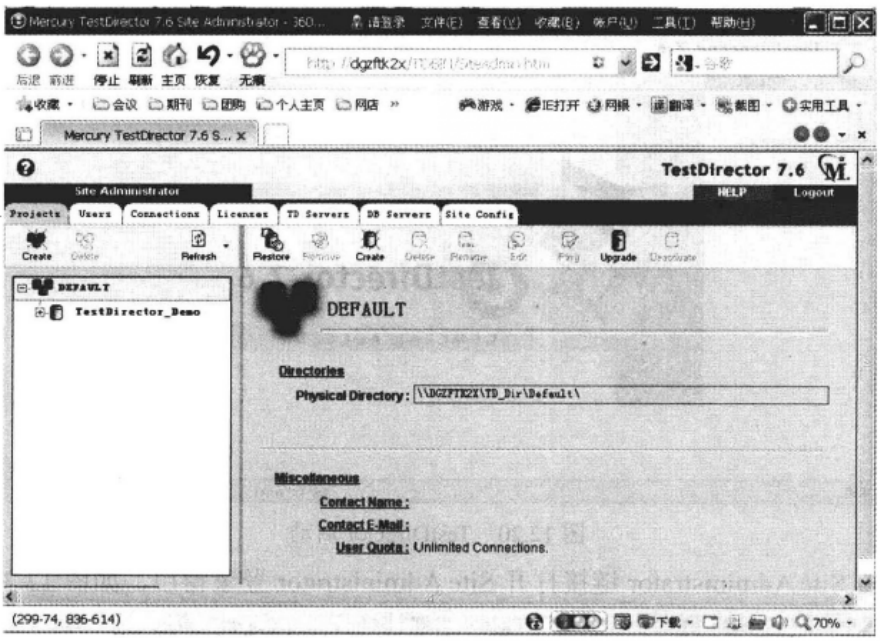


图 12.23 打开 Site Administrator 窗口

(5) 在 Site Administrator 窗口中点击 Projects 标签，标准版不能创建新域，只能在默认域下工作，点击 Create Project 按钮打开 Create Project 窗口，如图 12.24 所示。

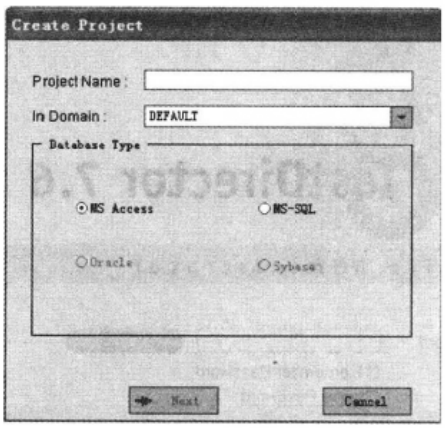


图 12.24 创建工程

(6) 在 Create Project 窗口中，如果使用的是企业版，可以选择项目的数据库类型，有 MS Access、MS-SQL、Oracle、Sybase 四种。输入项目名称，点击 Next 按钮，如图 12.25 所示。

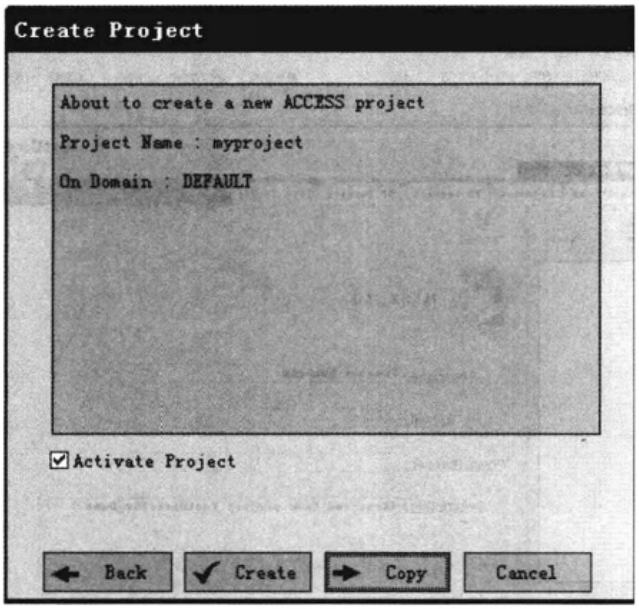


图 12.25　选择数据库

(7) 点击 Create 按钮创建新项目，内容为空，新项目被加入到项目列表中，可以看到图 12.26 中有关此项目的基本信息。

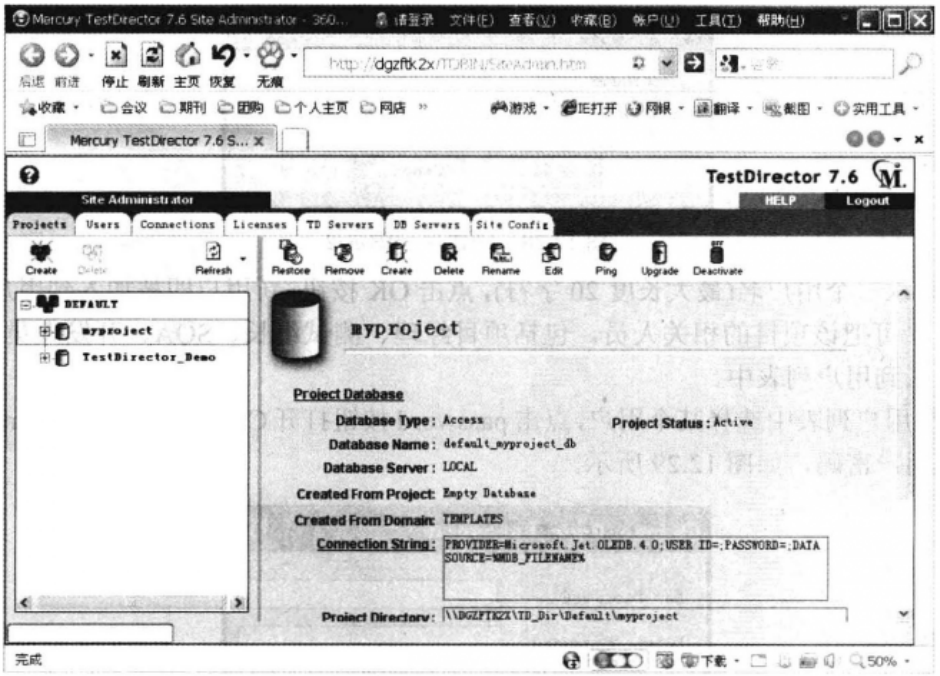


图 12.26　项目信息

2. 创建用户

(1) 在 Site Administrator 窗口中点击 Users 标签，如图 12.27 所示。

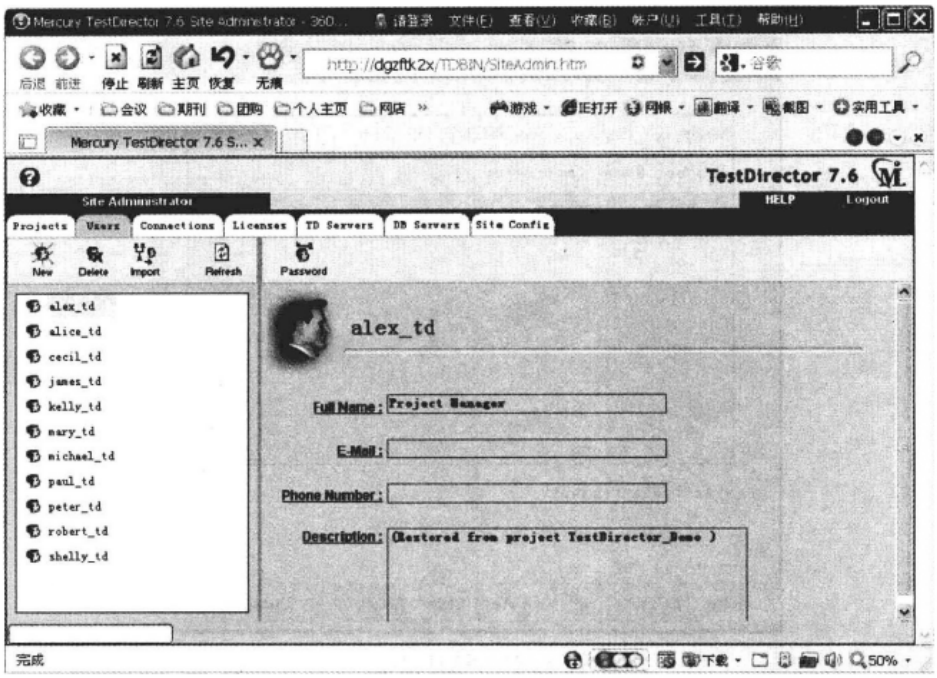


图 12.27 创建用户图示 1

(2) 点击 New User 按钮打开 Create User 窗口，如图 12.28 所示。

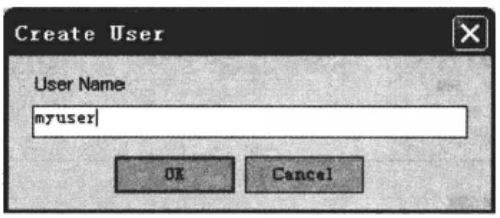


图 12.28 创建用户图示 2

(3) 输入一个用户名(最大长度 20 字符)，点击 OK 按钮，新用户即被加入到用户列表中。以此类推，可把该项目的相关人员，包括项目经理、测试组长、SQA、开发人员、测试人员陆续加入到用户列表中。

(4) 从用户列表中选择某个用户，点击 password 按钮打开 Change User Password 对话框，可以修改用户密码，如图 12.29 所示。

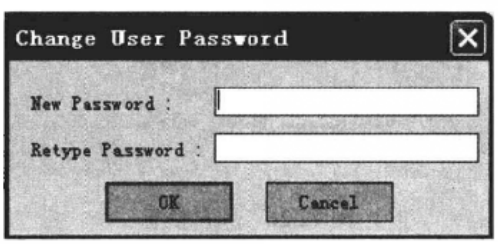


图 12.29 创建用户图示 3

(5) 对所有用户的初始密码设置完成后点击位于 Site Administrator 窗口右上方的 Logout 按钮退出。

3. 定制项目

(1) 打开浏览器输入 TestDirector URL(http://[server name]/[virtual directory name]/default.htm) 访问 TestDirector 选项窗口，如图 12.30 所示。

图 12.30　定制项目图示 1

(2) 点击 TestDirector 链接打开 TestDirector 登录窗口，如图 12.31 所示。

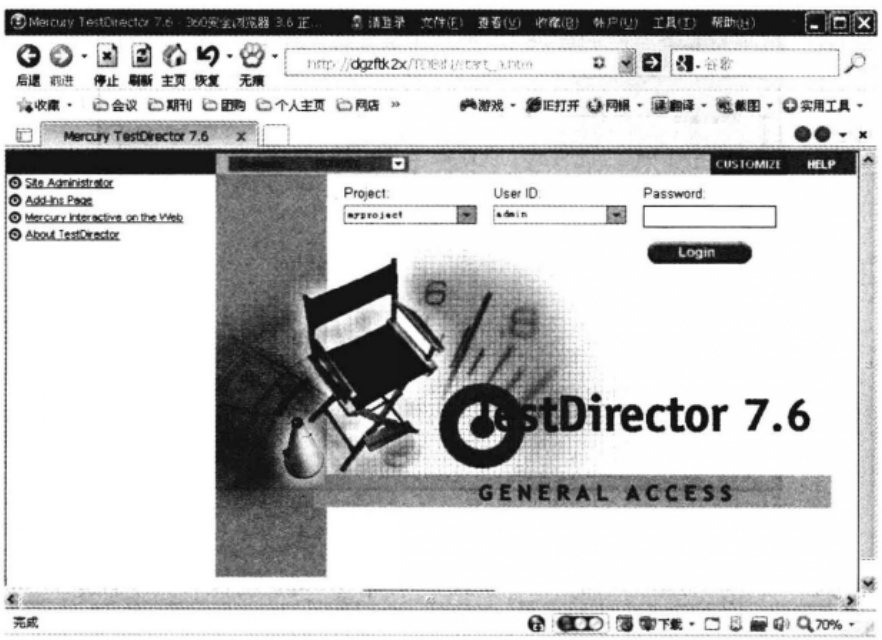


图 12.31　定制项目图示 2

(3) 点击位于窗口右上角的 CUSTOMIZE 按钮打开登录窗口，如图 12.32 所示。

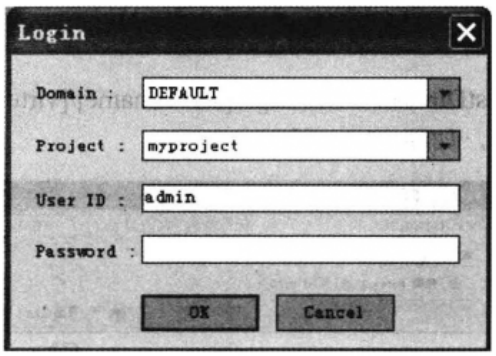


图 12.32 定制项目图示 3

(4) 选择刚才创建的新项目，以管理员登录，出现 Project Customization 窗口，如图 12.33 所示。

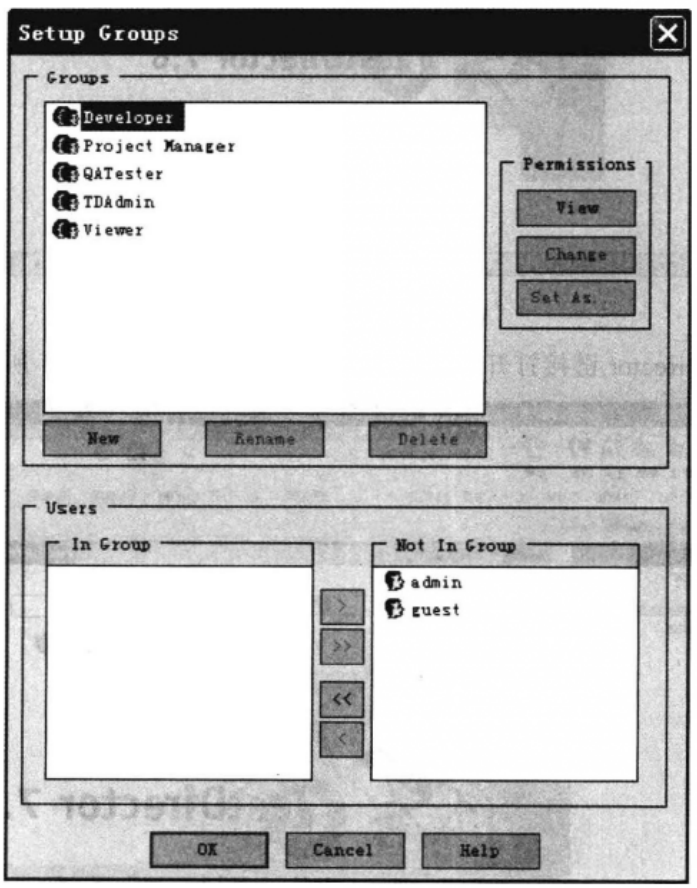


图 12.33 定制项目图示 4

(5) 在 Project Customization 窗口中，点击 SetupGroups 链接打开 Setup Groups 窗口，如图 12.34 所示。

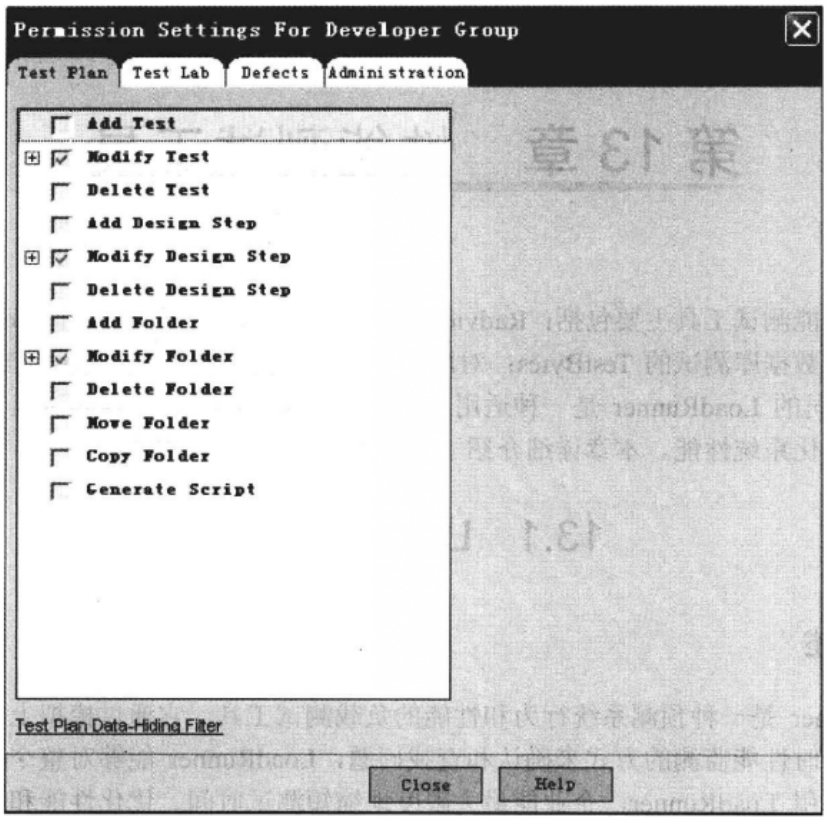


图 12.34　定制项目图示 5

(6) 在 Setup Groups 窗口中选择一个系统组，点击 View 按钮可以打开 Permission Settings 窗口以查看该组的访问许可权限。

(7) Permission Settings 窗口中有五个标签，分别针对测试需求模块、测试计划模块、测试库模块、缺陷模块、管理模块。需要注意的是，在测试计划模块、测试库模块、缺陷模块中，Permission Settings 窗口下方会有一个 Data-Hiding Filter 的链接，通过它可以实现更高级的定制，在此不再一一赘述。

第 13 章 性能测试工具

专业的性能测试工具主要包括：Radview 公司的 WebLoad；Microsoft 公司的 WebStress 等工具；针对数据库测试的 TestBytes；对应用性能进行优化的 EcoScope 等工具。Mercury Interactive 公司的 LoadRunner 是一种适用于各种体系架构的自动负载测试工具，它能预测系统行为并优化系统性能。本章详细介绍了 LoadRunner 的使用方法。

13.1 LoadRunner

13.1.1 综述

LoadRunner 是一种预测系统行为和性能的负载测试工具。它通过模拟上千万用户实施并发负载及实时性能监测的方式来确认和查找问题，LoadRunner 能够对整个企业架构进行测试。通过使用 LoadRunner，企业能最大限度地缩短测试时间、优化性能和加速应用系统的发布周期。

LoadRunner 具有如下特点。

1) 轻松创建虚拟用户

使用 LoadRunner 的 Virtual User Generator 能很简便地创立起系统负载。该引擎能够生成虚拟用户，以虚拟用户的方式模拟真实用户的业务操作行为。它先记录下业务流程(如下订单或机票预定)，然后将其转化为测试脚本。利用虚拟用户，可以在 Windows、UNIX 或 Linux 系统的机器上同时产生成千上万个用户访问，极大地减少负载测试所需的硬件和人力资源。

用 Virtual User Generator 建立测试脚本后，可以对其进行参数化操作，这一操作能让用户利用几套不同的实际发生数据来测试应用程序，从而反映出系统的负载能力。以一个订单输入过程为例，参数化操作可将记录中的固定数据，如订单号和客户名称，由可变值来代替。在这些变量内随意输入可能的订单号和客户名，来匹配多个实际用户的操作行为。

LoadRunner 通过 Data Wizard 来自动实现其测试数据的参数化。Data Wizard 直接连于数据库服务器，用户可以从中获取所需的数据(如定单号和用户名)并直接将其输入到测试脚本中。这样避免了人工处理数据的需要，Data Wizard 为用户节省了大量的时间。

2) 创建真实的负载

Virtual User 建立起后，用户需要设定负载方案、业务流程组合和虚拟用户数量。使用 LoadRunner 的 Controller，用户能很快组织起多用户的测试方案。Controlle 的 Rendezvous 功能提供了一个互动的环境，在其中既能建立起持续且循环的负载，又能管理和驱动负载测试方案。而且，用户可以利用它的日程计划服务来定义用户在什么时候访问系统以产生

负载，这样，就能将测试过程自动化。同样用户还可以用 Controller 来限定自己的负载方案，在这个方案中所有的用户同时执行一个动作，如登录到一个库存应用程序来模拟峰值负载的情况。另外，用户还能监测系统架构中各个组件的性能，包括服务器、数据库、网络设备等，来帮助客户决定系统的配置。

LoadRunner 通过它的 AutoLoad 技术，为用户提供了更多的测试灵活性。使用 AutoLoad，用户可以根据目前的用户人数事先设定测试目标，优化测试流程。例如，目标可以是确定应用系统承受的每秒点击数或每秒的交易量。

3) 定位性能问题

LoadRunner 内含有集成的实时监测器，在负载测试过程的任何时候，用户都可以观察到应用系统的运行性能。这些性能监测器为用户实时显示交易性能数据(如响应时间)和其它系统组件(包括 application server、web server、网路设备和数据库等)的实时性能。这样，用户就可以在测试过程中从客户和服务器的双方面评估这些系统组件的运行性能，从而更快地发现问题。

再者，利用 LoadRunner 的 ContentCheck TM，用户可以判断负载下的应用程序功能正常与否。ContentCheck 在 Virtual Users 运行时，检测应用程序的网络数据包内容，从中确定是否有错误内容传送出去。它的实时浏览器能帮助用户从终端用户角度观察程序性能状况。

4) 分析结果以精确定位问题所在

一旦测试完毕，LoadRunner 收集汇总所有的测试数据，并提供高级的分析和报告工具，以便迅速查找到性能问题并追溯原由。使用 LoadRunner 的 Web 交易细节监测器，用户可以了解到将所有的图像、框架和文本下载到每一网页上所需的时间。例如，这个交易细节分析机制能够分析是否因为一个大尺寸的图形文件或是第三方的数据组件造成应用系统运行速度减慢。另外，Web 交易细节监测器分解用于客户端、网络和服务器上端到端的反应时间，便于确认问题，定位查找真正出错的组件。例如，用户可以将网络延时进行分解，以判断 DNS 解析时间、连接服务器或 SSL 认证所花费的时间。通过使用 LoadRunner 的分析工具，用户能很快地查找到出错的位置和原因并作出相应的调整。

5) 重复测试保证系统发布的高性能

负载测试是一个重复过程，每次处理完一个出错情况，都需要对应用程序在相同的方案下再进行一次负载测试，以此检验所做的修正是否改善了运行性能。

6) Enterprise Java Beans 测试

LoadRunner 完全支持 EJB 的负载测试。这些基于 Java 的组件运行在应用服务器上，提供广泛的应用服务。通过测试这些组件，用户可以在应用程序开发的早期就确认并解决可能产生的问题。

利用 LoadRunner，用户可以很方便地了解系统的性能。Controller 允许重复执行与出错修改前相同的测试方案。它的基于 HTML 的报告为用户提供一个比较性能结果所需的基准，以此衡量在一段时间内，有多大程度的改进并确保应用成功。这些报告是基于 HTML 的文本的，用户可以将其公布于公司的内部网上，以便于随时查阅。

7) 最大化投资回报

所有 Mercury Interactive 公司的产品和服务都是集成设计的，能完全相容地一起运作。

由于它们具有相同的核心技术，因此来自于 LoadRunner 和 ActiveTest TM 的测试脚本在 Mercury Interactive 的负载测试服务项目中，可以被重复用于性能监测。借助 Mercury Interactive 的监测功能——Topaz TM 和 ActiveWatch TM，测试脚本可重复使用从而平衡投资收益。更重要的是，用户能为测试的前期布署和生产系统的监测提供一个完整的应用性能管理解决方案。

8) 支持无线应用协议

随着无线设备数量和种类的增多，测试计划需要同时满足传统的基于浏览器的用户和无线互联网设备，如手机和 PDA 的需求。LoadRunner 支持两项最广泛使用的协议：WAP 协议和 I-mode 协议。此外，通过负载测试系统整体架构，LoadRunner 能让用户只需记录一次脚本，就可完全检测上述这些无线互联网系统。

9) 支持 Media Stream 应用

LoadRunner 还能支持 Media Stream 应用。为了保证终端用户得到良好的操作体验和高质量 Media Stream，用户需要检测 Media Stream 应用程序。使用 LoadRunner，可以记录和重放任何流行的多媒体数据流格式来诊断系统的性能问题，查找原由，分析数据的质量。

10) 完整的企业应用环境的支持

LoadRunner 支持广泛的协议，可以测试各种 IT 基础架构。

13.1.2 测试示例

安装 LoadRunner 8.02 版后，会自动附带 Flight Reservation(航班预订)软件作为测试范例。

下面通过 LoadRunner 自带的飞机订票网站示例，讲解 LoadRunner 的使用过程。在使用该示例之前需要保证如下条件：

1. 确保示例 Web 服务器正在运行

安装和重新启动 LoadRunner 后，Web 服务器将自动启动。如果再次重新启动系统后，该服务器没有运行，请依次选择“开始”→“程序”→“Mercury LoadRunner”→“示例”→“Web”→“启动 Web 服务器”。

2. 打开 Mercury Tours 应用程序

选择“开始”→“程序”→“Mercury LoadRunner”→“示例”→“Web”→“Mercury Web Tours 应用程序”，将打开浏览器，其中显示 Mercury Tours 的起始页。

3. 登录

键入下列信息：

成员名：jojo；密码：bean。

单击左窗格中的“登录”，将显示 Mercury Tours 的欢迎页。

4. 预订航班

单击左窗格中的“航班”，将打开“查找航班”页。将目的地更改为洛杉矶，单击“继续”。

5. 结束 Mercury Tours 会话

单击“注销”进行注销。

LoadRunner 测试过程如下所示。

1. 使用 VuGen 创建脚本

创建负载测试的第一步是使用 VuGen 录制典型最终用户的业务流程。VuGen 采用录制并回放机制。当用户在应用程序中按照业务流程操作时，VuGen 将这些操作录制到自动脚本中，以便作为负载测试的基础。

在此部分中，将录制旅行代理为一位乘客预订从丹佛到洛杉矶的航班的流程。

1) 准备录制

首先打开 VuGen 并创建一个空白脚本。

(1) 启动 LoadRunner。选择“开始”|“程序”|“Mercury LoadRunner”|“LoadRunner”命令，将打开“Mercury LoadRunner”窗口，如图 13.1 所示。

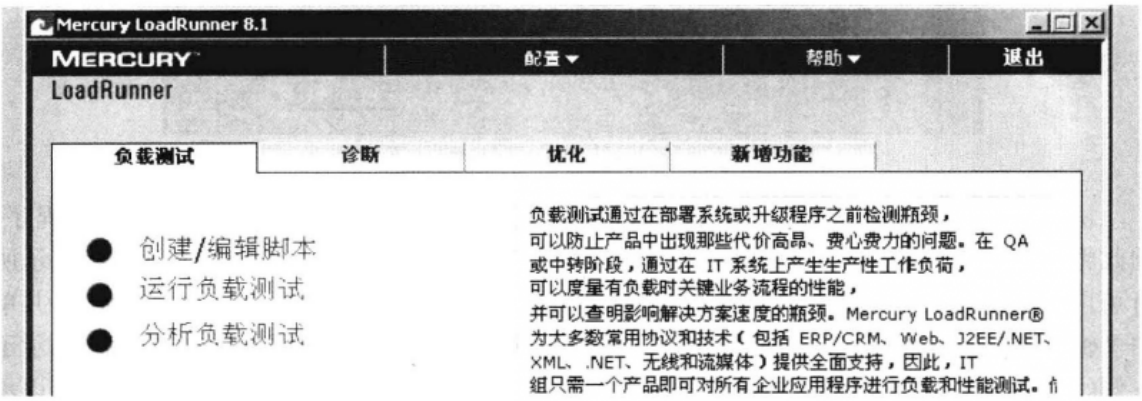


图 13.1　“Mercury LoadRunner”窗口

(2) 打开 VuGen。在“负载测试”选项卡中，单击“创建/编辑脚本”命令，将打开 VuGen 开始页，如图 13.2 所示。

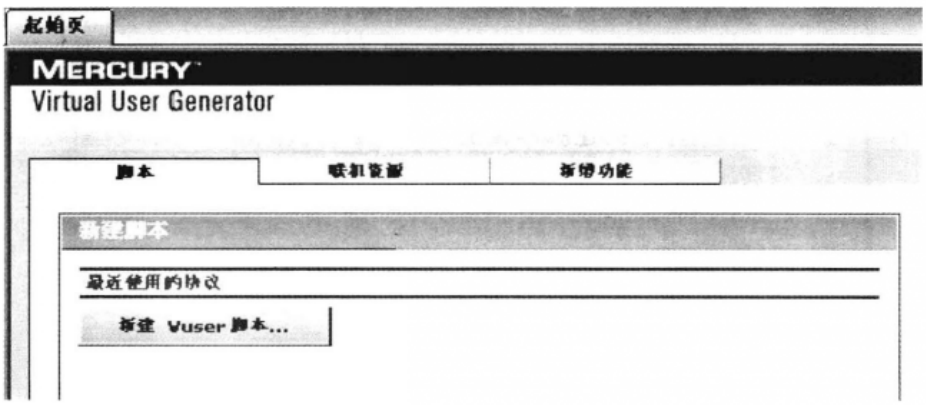


图 13.2　VuGen 开始页

(3) 创建一个空白 Web 脚本。在 VuGen 开始页中的“脚本”选项卡中，单击“新建 Vuser 脚本”命令，将打开“新建虚拟用户”对话框，并显示“新建单协议脚本”屏幕，如图 13.3 所示。

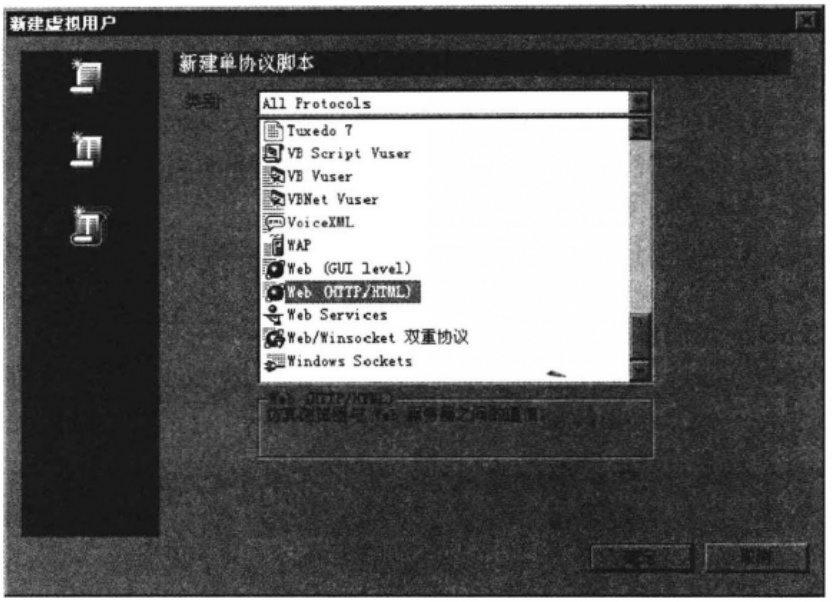


图 13.3 “新建虚拟用户”对话框

协议是客户端用来与系统后端进行通信的语言。Mercury Tours 是基于 Web 的应用程序，因此将创建一个 Web 虚拟用户脚本。确保“类别”类型为“所有协议”，VuGen 将显示所有可用于单协议脚本的协议列表。向下滚动查看该列表，选择“Web(HTTP/HTML)”并单击“确定”，创建一个空白 Web 脚本。空脚本以 VuGen 向导模式打开，且任务窗格显示于左侧(如果未显示任务窗格，请单击工具栏上的“任务”按钮)。VuGen 向导将指导用户逐步完成创建脚本，然后根据实际的测试环境进行相应的修改。

任务窗格列出了脚本创建过程中的每个步骤或任务。在用户逐步完成每一步操作的过程中，VuGen 会在窗口的主区域显示详细的说明和准则，如图 13.4 所示。

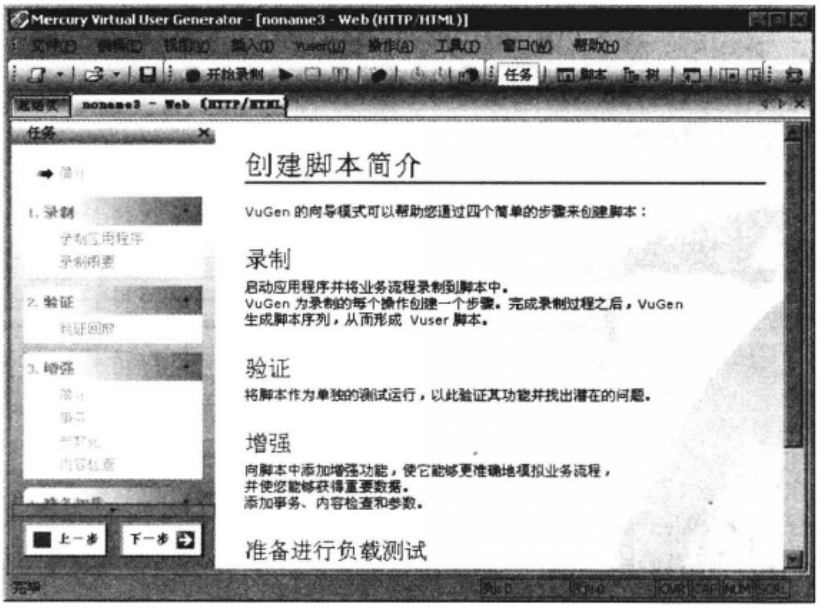


图 13.4 任务窗格

(4) 使用 VuGen 向导录制业务流程。

具体录制过程如下：

① 在 Mercury Tours 网站上开始录制。在任务窗格中，单击步骤 1 中的“录制应用程序”，单击说明窗格底部的“开始录制”按钮 开始录制，将打开“开始录制”对话框，如图 13.5 所示。

图 13.5　“开始录制”对话框

在“URL 地址”框中，键入 http://localhost:1080/MercuryWebTours/。在“录制到操作”框中，选择“操作”，单击“确定”按钮，将打开一个新的 Web 浏览器，并显示 Mercury Tours 站点。(如果在打开站点时出现错误，请确保 Web 服务器正在运行。要启动服务器，请选择“开始”|“程序”|“Mercury LoadRunner”|“示例”|“Web”|“启动 Web 服务器”。)

此时将打开浮动的“录制”工具栏，如图 13.6 所示。

图 13.6　“录制”工具栏

② 登录到 Mercury Tours 网站。在“成员姓名”框中输入 jojo，在“密码”框中输入 bean。单击“登录”，将打开欢迎页面。

③ 输入航班详细信息。单击“航班”，将打开“查找航班”页面。

◆ 出发城市：丹佛(默认设置)。

◆ 出发日期：保持默认设置不变(当前日期)。

◆ 到达城市：洛杉矶。

◆ 返回日期：保持默认设置不变(第二天的日期)。保持其余的默认设置不变，然后单击“继续”，将打开“搜索结果”页面。

④ 选择航班。单击“继续”接受默认航班选择，将打开“付费详细信息”页面。

⑤ 输入付费信息并预订航班。在“信用卡”框中输入 12345678，并在“过期日期”框中键入日期。单击“继续”，将打开“发票”页面，并显示订单发票。

⑥ 查看路线。在左窗格中单击 “路线”，将打开“路线”页面。

⑦ 在左窗格中单击“注销”。

⑧ 单击浮动工具栏上的“停止”以停止录制过程。

一旦生成了 Vuser 脚本，Vuser 向导将自动前进到任务窗格中的下一步，并显示包含协议信息以及在会话期间创建的一系列操作的录制概要。对于录制期间执行的每个步骤，

VuGen 都生成一个快照(即录制期间各窗口的图片)。这些快照的缩略图显示在右侧窗格中。

⑨ 选择“文件”|“保存”，或单击“保存”。在“文件名”框中键入 basic_tutorial 并单击“保存”。VuGen 将把该文件保存在 LoadRunner 脚本文件夹中，并在标题栏中显示该测试名称。

现在，您可以查看在 VuGen 中录制的脚本。可以在树视图或脚本视图中查看脚本。树视图是基于图标的视图，其中将 Vuser 的操作作为步骤列出；而脚本视图是基于文本的视图，其中将 Vuser 的操作作为函数列出。

● 树视图

要在树视图中查看脚本，请选择“查看”|“树视图”或单击“树视图”按钮。对于录制期间执行的每个步骤，VuGen 都在测试树中生成了一个图标和一个标题。

在树视图中，将用户的操作作为脚本步骤列出。大多数步骤都附带相应的录制快照，如图 13.7 所示。

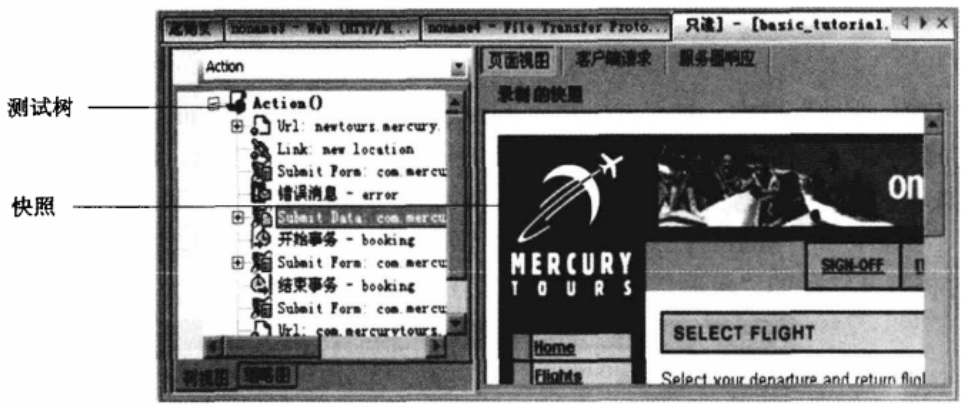


图 13.7 树视图

● 脚本视图

脚本视图是基于文本的视图，其中将 Vuser 的操作作为 API 函数列出。要在脚本视图中查看脚本，请选择“查看”|“脚本视图”或单击“脚本视图”按钮。

在图 13.8 所示的脚本视图中，VuGen 在编辑器中通过彩色编码函数及其参数值显示脚本。可以直接在此窗口键入 C 代码或 LoadRunner API 函数以及控制流语句。

图 13.8 脚本视图

2) 回放脚本

完成录制后，可以回放脚本，以便验证它是否准确地模拟了所录制的操作。

(1) 确保显示了任务窗格(如果没有，请单击工具栏中的“任务”按钮)。

单击任务窗格中的“验证回放”，然后单击说明窗格底部的“开始回放”按钮。

(2) 如果打开了“选择结果目录”对话框，询问要存储结果目录的位置，请接受默认名称并单击“确定”。

一段时间之后，VuGen 开始运行脚本。脚本停止运行后，就可以在向导中查看回放概要了。

(3) 单击任务窗格中的“验证回放”查看上次回放概要。

上次回放概要列出了可能检测到的所有错误并显示录制和回放快照的缩略图。通过比较快照，可以了解录制和回放之间的差异。

可以使用“运行时设置”模拟各种不同类型的用户行为。例如，可以模拟一个对服务器立即做出响应的用户，也可以模拟一个在做出响应之前先停下来思考的用户。

3) 增强脚本

准备负载测试过程时，LoadRunner 允许增强脚本以使其更好地反映真实情况。例如，可以在脚本中插入名为内容检查的步骤，以验证某些特定内容是否显示在返回页上。可以修改脚本来模拟多用户行为，也可以指示 VuGen 度量特定的业务流程。

准备要部署的应用程序时，需要度量特定业务流程的持续时间，如登录、预订航班等所花费的时间。这些业务流程通常由脚本中的一个或多个步骤或操作构成。

在 LoadRunner 中，可以通过将想要度量的操作标记为事务来指定一系列操作。通过在脚本中插入一个事务以度量用户查找和确认航班所花费的时间。

(1) 打开事务创建向导。

首先，确保显示了任务窗格(如果没有，请单击 “任务”按钮)，再在任务窗格的“增强功能”标题下单击“事务”，将打开事务创建向导。事务创建向导显示脚本中不同步骤的缩略图。

单击“新建事务”按钮。现在，可以拖动事务标记并将其放置在脚本中的指定点，向导会提示插入事务的起始点，如图 13.9 所示。

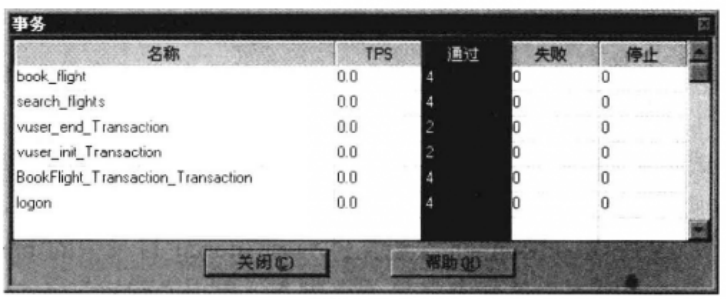


图 13.9　新建事务

(2) 插入开始事务标记和结束事务标记。

使用鼠标将标记放置到标题为搜索航班按钮的第三个缩略图之前并单击，向导会提示插入结束点。

使用鼠标将标记放置到标题为 reservations.pl_2 的第五个缩略图之后并单击。

(3) 指定事务的名称。

向导会提示用户输入事务的名称。键入 find_confirm_flight，然后单击“Enter”，可以通过将标记拖动到脚本中的其它点来调整事务的起始点或结束点，也可以单击开始事务标记上的现有名称并键入新名称来重命名事务。

2. 使用 Controller 设计场景

使用 Controller，可以将应用程序性能测试需求划分为多个场景，场景定义每个测试会话中发生的事件。例如，一个场景可以定义和控制模拟的用户数、用户执行的操作以及用户运行其模拟时所用的计算机。

1) 创建场景

此部分的目标是创建一个场景，用来模拟 10 个旅行代理同时登录系统、搜索航班、购买机票、查看路线和注销系统。

(1) 打开 Mercury LoadRunner。选择“开始”|“程序”|“Mercury LoadRunner”|“LoadRunner”，将打开“Mercury LoadRunner Launcher”窗口。

(2) 打开 Controller。在“负载测试”选项卡中，单击“运行负载测试”，将打开 LoadRunner Controller。默认情况下，Controller 打开时将显示“新建场景”对话框，如图 13.10 所示。

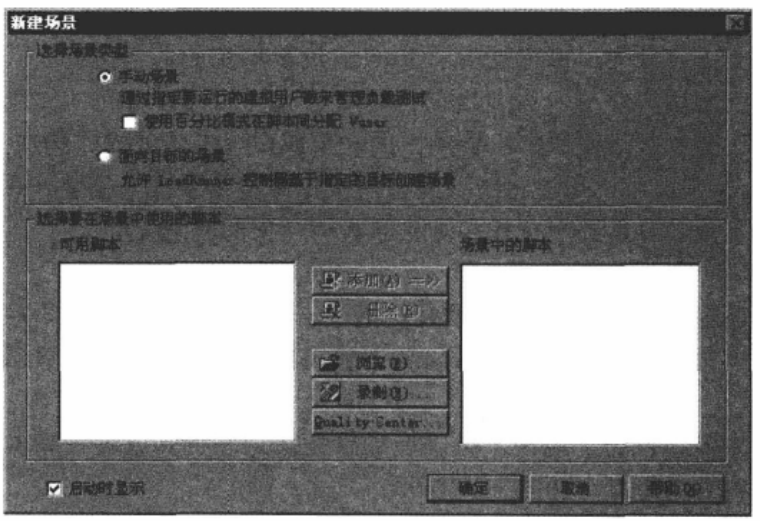


图 13.10 “新建场景”对话框

(3) 选择场景类型。选择“手动场景”。Controller 允许用户选择各种不同的场景类型(例如面向目标的场景)。

(4) 向负载测试添加脚本。为配合快速入门，提供了一个与您创建的脚本相似的脚本。建议您使用该示例脚本。

单击“浏览”，找到<LoadRunner 安装文件夹>\Tutorial 目录中的 basic_script。“可用脚本”部分和“场景中的脚本”部分中将显示该脚本。单击“确定”，LoadRunner Controller 的“设计”选项卡中将显示所创建的场景。

2) 设计场景

Controller 窗口的“设计”选项卡包含“场景计划”和 “场景组”两个主要部分，如图 13.11 所示。

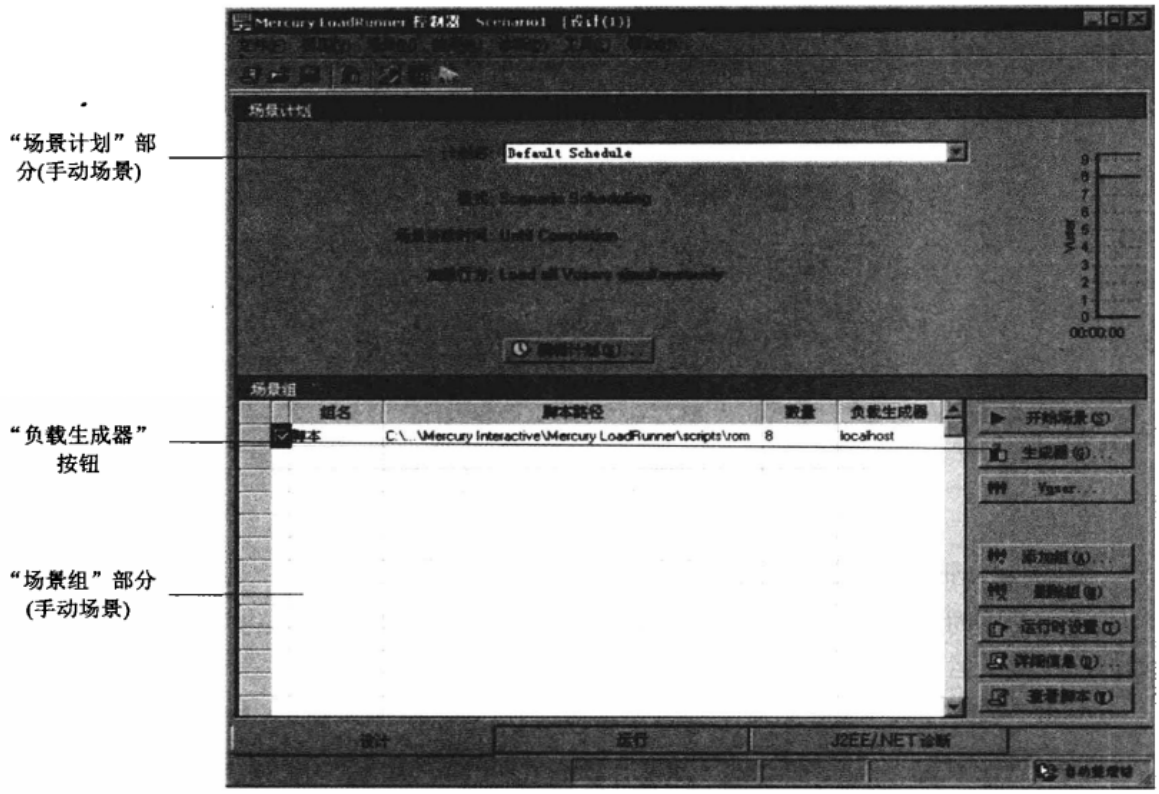


图 13.11　设计场景

场景计划：在“场景计划”部分，可以设置负载行为以准确描绘用户行为。可以确定将负载应用于应用程序的频率、负载测试的持续时间和停止负载的方式等。

场景组：可以在“场景组”部分配置 Vuser 组。可以创建不同的组来代表系统的典型用户。您可以定义这些典型用户运行的操作、运行的 Vuser 数以及 Vuser 运行时所用的计算机。

负载生成器：负载生成器是通过运行 Vuser 在应用程序中创建负载的计算机。可以使用多台负载生成器计算机，并在每台计算机上创建多个虚拟用户。

3) 计划场景

由于通常不会有多个典型用户恰好同时登录和注销系统，因此 LoadRunner 的 Controller 计划生成器允许您建立较准确描绘典型用户行为的场景计划。例如，在创建手动场景后，设置场景的持续时间或选择在场景中逐渐运行和停止 Vuser。通过使用 Controller 计划生成器更改默认负载设置。

(1) 更改场景计划默认设置。单击“编辑计划”按钮，将打开计划生成器，如图 13.12 所示。

(2) 指定逐渐开始。在“加压”选项卡中，将设置更改为“每 15 秒开始 2 个 Vuser”。

(3) 计划持续时间。在“持续时间”选项卡中，将设置更改为“在加压完成之后运行 3 分钟”。

(4) 计划逐渐关闭。在“减压”选项卡中，将设置更改为“每 30 秒停止 5 个 Vuser”，单击“确定”。

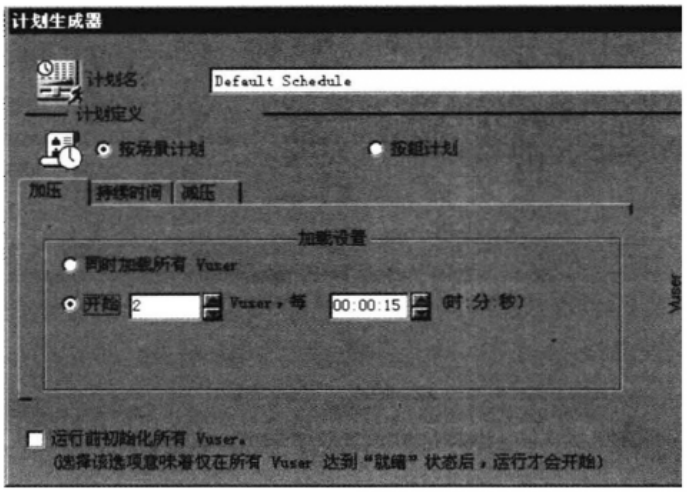


图 13.12 计划生成器

3. 使用 Controller 运行场景

通过前面的操作，已经设计了负载测试场景，接下来可以运行该测试并观察应用程序如何在负载下运行。

单击“运行”选项卡，打开“运行”视图，如图 13.13 所示。

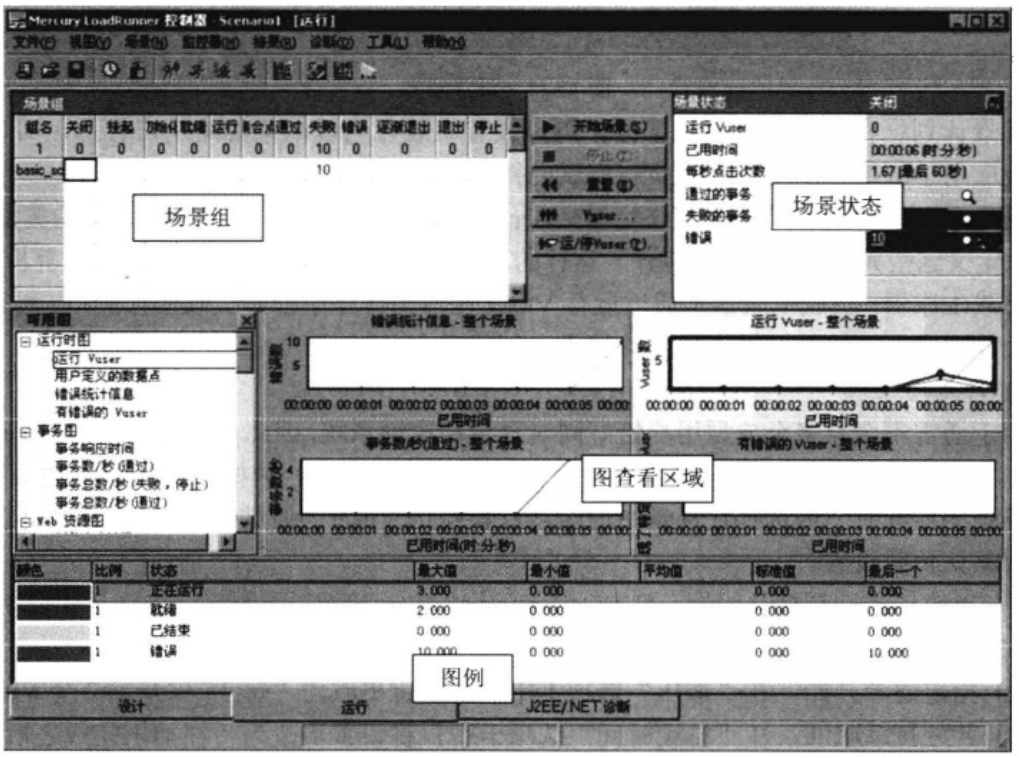


图 13.13 “运行”视图

“运行”视图包含五个主要部分：

- 场景组：位于左上窗格中，在其中可以查看场景组中的 Vuser 的状态。使用该窗格右

侧的按钮可以启动、停止和重置场景，查看单个 Vuser 的状态，并且可以手动添加更多的 Vuser，从而增加场景运行期间应用程序上的负载。

• 场景状态：位于右上窗格中，在其中可以查看负载测试的概要，包括正在运行的 Vuser 数以及每个 Vuser 操作的状态。

• 可用图树：位于中部左侧窗格中，在其中可以查看 LoadRunner 图列表。若要打开图，请在该树中选择一个图，然后将其拖动到图查看区域中。

• 图查看区域：位于中部右侧窗格中，在其中可以自定义查看 1～8 个图(“视图”|“查看图”)。

• 图例：位于底部窗格中，在其中可以查看选定图中的数据。选择一行，图中的相应行将突出显示，反之亦然。

(1) 开始场景。单击“开始场景”按钮 开始场景(S) 或选择“开始”|“场景”以开始运行测试。场景运行大约 5 分钟。

(2) 通过 Controller 的联机图监控性能。测试运行时，可以通过 LoadRunner 的一组集成监控器查看应用程序如何实时执行以及潜在瓶颈所在位置，在 Controller 的联机图上查看监控器收集的性能数据。

4. 分析场景结果

具体操作如下所述：LoadRunner Analysis 将性能数据收集到详细的图和报告中。使用这些图和报告，可以轻松地确定和标识应用程序中的瓶颈以及提高系统性能所需的改进。

(1) 从 Controller 的菜单中选择“工具”|“Analysis”或选择“开始”|“程序”|“Mercury LoadRunner”|“应用程序”|“Analysis”来打开 LoadRunner Analysis。

(2) 在 Analysis 窗口中，选择“文件”|“打开”，将打开“打开现有 Analysis 会话文件”对话框。

(3) 在<LoadRunner 安装目录>\Tutorial 文件夹中，选择 analysis_session 并单击“打开”，Analysis 将在 Analysis 窗口中打开该会话文件。

• 概要报告

LoadRunner Analysis 打开时显示概要报告。概要报告提供有关场景运行的一般信息。在报告的统计信息概要中，可以了解到测试中运行的用户数，并可查看其它统计信息(如总/平均吞吐量和总/平均点击次数)。

• 查看图

Analysis 窗口左窗格的图树中列出了已经打开可供查看的图，这些图显示在 Analysis 窗口右窗格的图查看区域中，可以在该窗口底部窗格的图例中查看选定图中的数据。

• 平均事务响应时间

通过平均事务响应时间图，可以查看在场景运行期间有问题的事务行为。

(1) 在图树中单击“平均事务响应时间”，平均事务响应时间图即显示在图查看区域中。

(2) 在图例中，单击 check_itinerary，check_itinerary 事务即突出显示在该图以及图下方的图例中，如图 13.14 所示。

请注意，与图底部平均响应时间相对稳定的其它事务相比，check_itinerary 事务的平均响应时间的波动非常大。

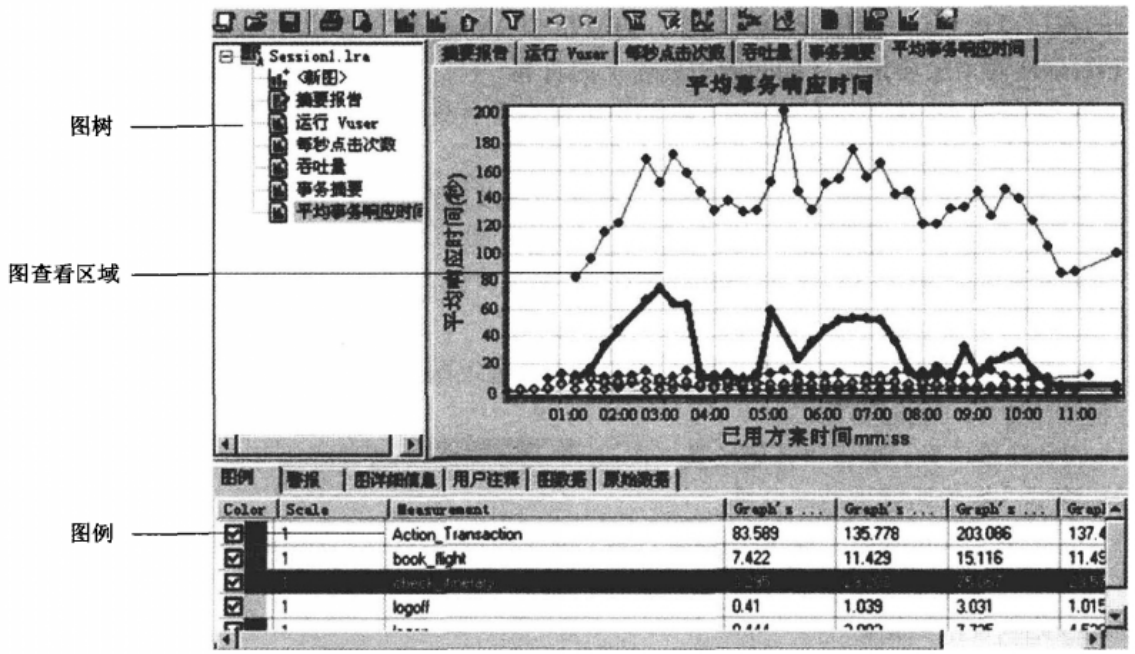


图 13.14 选中 check_itinerary 事务

• 合并图和关联图

将两个图联系起来，就会看到一个图的数据会对另一个图的数据产生影响，这称为将两个图关联。例如，可以将正在运行的 Vuser 图和平均事务响应时间图相关联，来了解大量的 Vuser 对事务的平均响应时间产生的影响。

(1) 在图树中单击“正在运行的 Vuser”，查看正在运行的 Vuser 图。

(2) 右键单击正在运行的 Vuser 图并选择“合并图”。

(3) 在“选择要合并的图”列表中，选择“平均事务响应时间”。

(4) 在“选择合并类型”区域中，选择“关联”，然后单击“确定”，此时正在运行的 Vuser 图和平均事务响应时间图即显示在一个图(该图显示在图查看区域中)中，如图 13.15 所示。

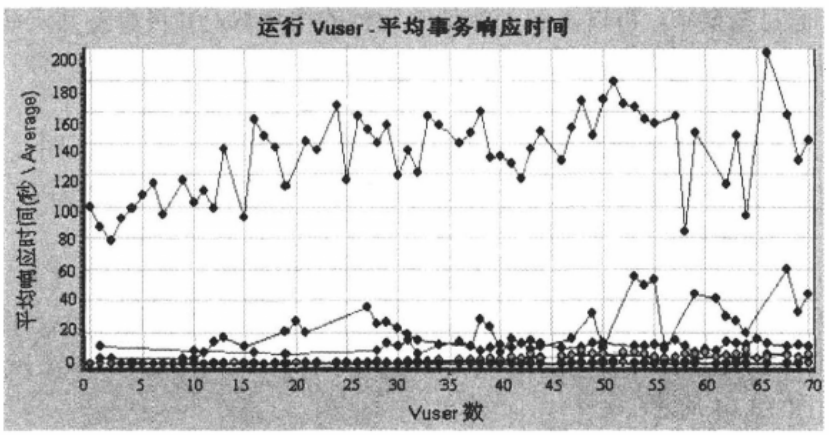


图 13.15 合并图

另一个 Analysis 工具自动关联用来合并所有包含可能对给定事务产生影响的数据的图。

事务与每个元素的关联都显示出来，这样，就可以推断哪些元素对给定事务的影响最大。

- 筛选图数据和排序图数据

通过对图数据进行筛选，以显示特定场景段的较少事务；对图数据进行排序，以更多相关方式来显示数据。例如，您可以对平均事务响应时间图进行筛选以仅显示 check_itinerary 事务。

(1) 在图树中单击“平均事务响应时间”，打开该图。

(2) 右键单击该图并选择“设置筛选器/分组方式”。

(3) 在“事务名称”框中选择 check_itinerary 并单击“确定”，筛选的图仅显示 check_itinerary 事务并隐藏所有其它事务。

5. 发布 HTML 报告和 Microsoft Word 报告

通过采用 HTML 报告或 Microsoft Word 报告的形式发布 Analysis 会话的结果。HTML 报告可以在任何浏览器中打开和查看。Word 报告比 HTML 报告更全面，它可以包含有关场景的一般信息，可以对报告进行格式设置以包含公司名称、徽标以及作者的详细信息等。

第 14 章 缺陷跟踪管理工具

缺陷跟踪管理是测试工作的一个重要部分，必须具有良好的工具支持才能充分发挥其作用，本章介绍两种广泛应用的缺陷跟踪管理工具：Bugzilla 和 Jira。

14.1 缺陷跟踪管理工具——Bugzilla

Bugzilla 是由 Mozilla 公司提供的一个开源的缺陷跟踪工具，在全世界拥有大量用户。作为一个产品缺陷的记录及跟踪工具，它能够为软件组织建立一个完善的缺陷跟踪体系，包括报告缺陷、查询缺陷记录并产生报表、处理解决缺陷、管理员系统初始化和设置等。

Bugzilla 主要为 Unix/Linux 系统设计，也可以运行在 Windows 系统上。可从网站 http://www.bugzilla.org 下载 Bugzilla 的软件包。Bugzilla 的运行需要数据库系统 MySQL、Perl 语言包和 Web 服务器(如 Tomcat)的支持。

14.1.1 Bugzilla 的特点

Bugzilla 具有以下特点：

(1) 基于 Web 方式，安装简单，运行方便快捷，管理安全。

(2) 提供强大的查询匹配能力，能根据各种条件组合进行缺陷查询，并能够记忆搜索条件。

(3) 缺陷从最初的报告到最后的关闭都有详细的操作记录，确保缺陷不会被忽略，并允许用户在检查缺陷状态时获取历史记录。

(4) 自带基于当前数据库的报表生成功能，主要生成两类图表：基于表格的视图和图形视图(条形图、线图、饼状图)。

(5) 具有灵活和完善的权限管理功能，管理员可以根据需要定义由个人或者小组构成的访问组。可以定义一组特殊用户，他们所发表的评论和附件只能被组内成员访问。

(6) 模型化的验证模块，方便用户添加所需系统验证。Bugzilla 内置了对 MySQL 和 LDAP 授权验证的支持。

(7) 可本地化配置：管理员可以根据用户所在地域来配置使用本地的字体进行页面显示。

(8) 评论回复链接：对缺陷的评论提供直接的页面链接，帮助复查人员评审缺陷。

14.1.2 Bugzilla 的缺陷处理流程

(1) 测试人员或开发人员发现 Bug 后，判断该 Bug 属于哪个模块的问题，填写 Bug 报告后，通过 E-mail 通知项目组长或直接通知开发者。

(2) 项目组长根据具体情况，重新分配(reassigned)给 Bug 所属的开发者。

(3) 开发者收到 E-mail 信息后，判断是否为自己的修改范围。

◆ 若不是，重新分配(reassigned)给项目组长或应该负责的开发者。

◆ 若是，进行处理，resolved 并给出解决方法。(可创建补丁附件及补充说明)

(4) 测试人员查询开发者已修改的 Bug，进行重新测试。(可创建 test case 附件)

◆ 经验证无误后，修改状态为 Verified。待整个产品发布后，修改为 Closed。

◆ 若还有问题，Reopened，状态重新变为“New”，并发邮件通知相关人员。

(5) 如果这个 Bug 一周内一直没被处理过，Bugzilla 就会一直通过 E-mail 通知它的属主，直到 Bug 被处理。

14.1.3　Bugzilla 的基本操作

1. 新建一个用户账号

(1) 点击“Open a new Bugzilla account”链接，输入用户的 E-mail 地址，然后点击“Create Account”。

(2) 稍候，用户会收到一封电子邮件，邮件中包含用户的登录账号(与 E-mail 相同)和口令，这个口令是 Bugzilla 系统随机生成的，用户可以根据需要进行变更。

随后可使用该账号登录 Bugzilla。

2. 报告 Bug

报告一个新的 Bug 时，需在图 14.1 所示的界面中输入 Bug 的相关信息。

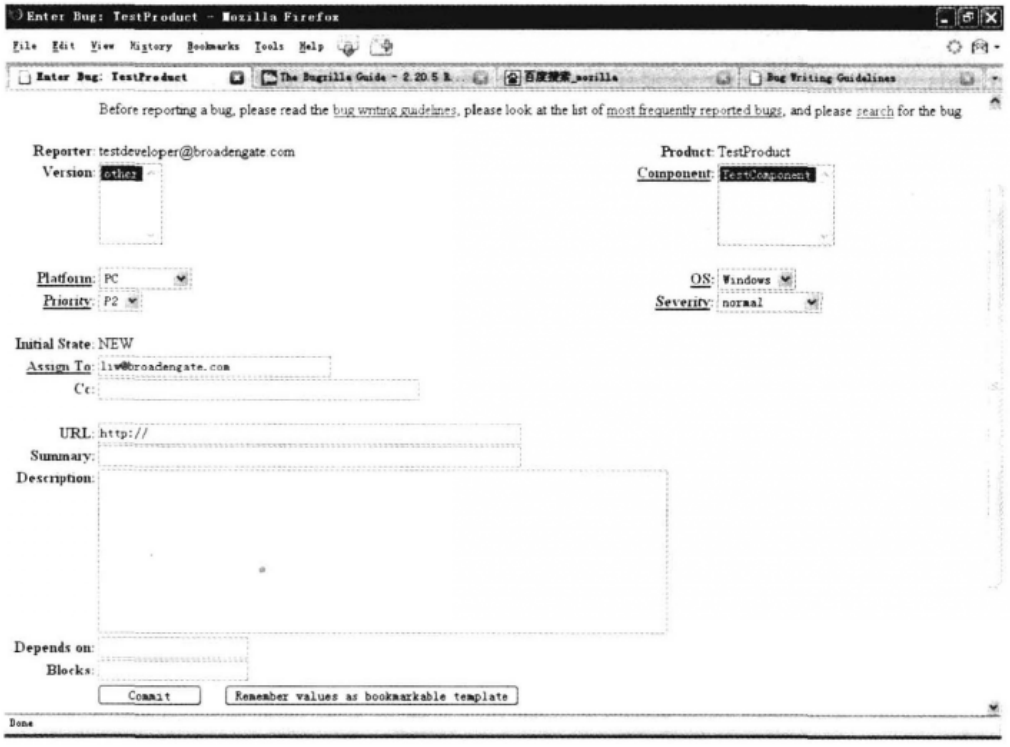


图 14.1　报告一个新的 Bug

各输入信息含义如下：

Priority 指缺陷的优先级。

Initial State 指 Bug 的初始状态，在 Bug 的处理流程中，有以下几种状态：

◆ Unconfirmed：待确认的。

◆ New：新提交的。

◆ Assigned：已分配的，指已分配给相关人员进行修复。

◆ Reopened：因问题未解决而重新打开的。

◆ Resolved：缺陷已修复而待返测的。

◆ Verified：已经过返测而待归档的。

◆ Closed：已归档的。

Assign To 指将缺陷分配给哪一个人员进行处理。

Cc 指可同时接到 Bug 报告通知的人员，如果为多人，需用“,”隔开。

URL 是一个超链接地址，引导处理人找到与缺陷报告相关联的信息。

Summary 是概述部分，填写时应保证处理人在阅读时能够清楚提交者在进行什么操作的时候发现了什么问题。如果是通用组件部分的测试，则必须将这一通用组件对应的功能名称写入概述中，以便今后查询。

在 Description 中要详细说明下列情况：

◆ 发现问题的步骤。

◆ 执行上述步骤后出现的情况。

◆ 期望应出现的正确结果。

Depends on 指如果该 Bug 必须在其它 Bug 修改以后才能够修改，则在此项目后填写那个 Bug 的编号。

Blocks 指如果该 Bug 影响其它 Bug 的修改，则在此项目后填写被影响的 Bug 编号。

Component 指 Bug 所在的软件模块。

Severity 指缺陷的严重程度，可选择以下选项：

◆ Blocker：阻碍开发和/或测试工作。

◆ Critical：关键性缺陷，如死机、丢失数据、内存溢出等。

◆ Major：较大的功能缺陷。

◆ Normal：普通的功能缺陷。

◆ Minor：较轻的功能缺陷。

◆ Trivial：产品外观上的问题或不影响使用的问题，如菜单或对话框中的文字拼写问题等。

填写了以上信息后，单击“Commit”按钮，提交 Bug，Bugzilla 会自动发送邮件通知负责处理缺陷的人员。

3．Bug 的处理

Bug 的属主(负责修复缺陷的人员)进行缺陷修复的操作方法如下：

登入 Bugzilla 系统后，点击浏览器下方的“Saved Searches: My Bugs”按钮进入 Bug 管

理界面，选择要修复的 Bug，修复完毕后，给出解决方式并填写 Additional Comments，还可创建附件(如更改提交单)，如图 14.2 所示。

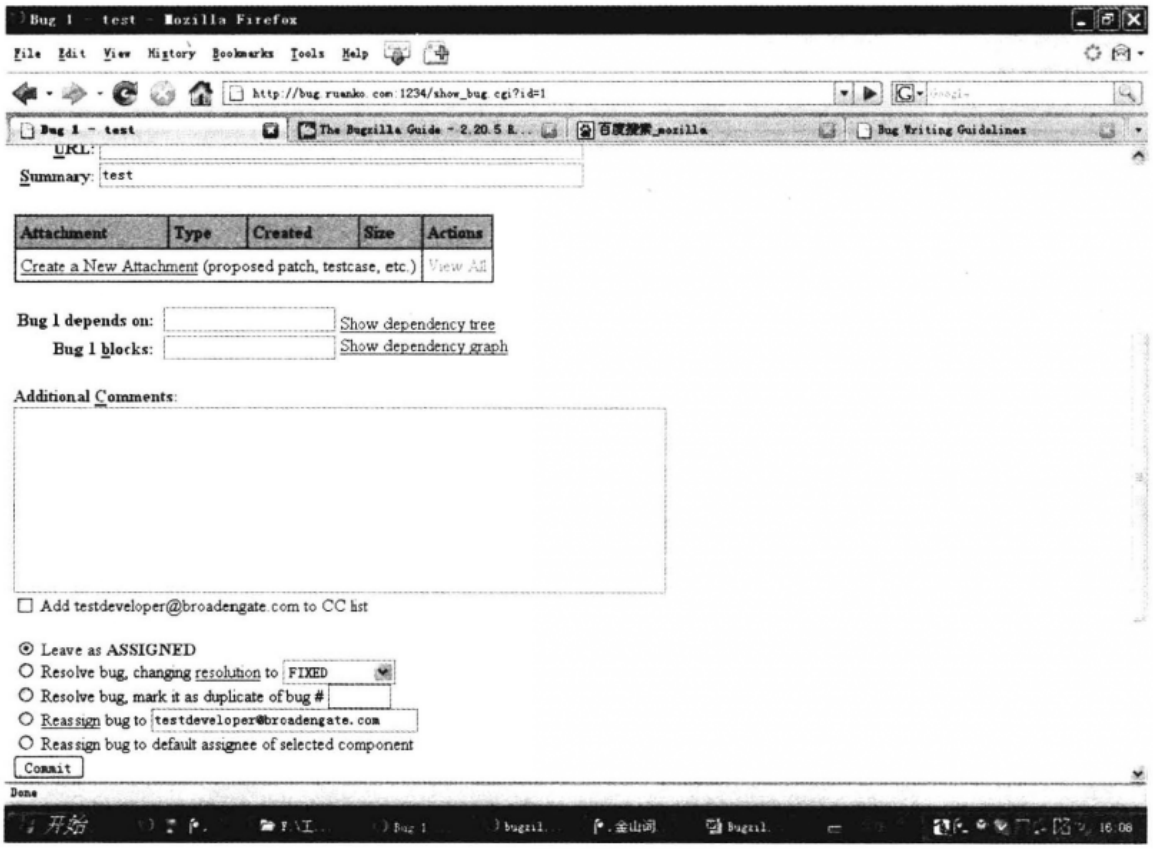


图 14.2　处理 Bug

Bug 的解决方式有以下几种：

◆ FIXED：问题已经修复。

◆ INVALID：描述的问题不是一个 Bug。

◆ WONTFIX：描述的问题将永远不会被修复。

◆ LATER：描述的问题将不会在产品的这个版本中解决。

◆ DUPLICATE：描述的问题与以前的某个 Bug 重复。

◆ WORKSFORME：无法重现 Bug。

如果负责处理 Bug 的人员发现此 Bug 不属于自己的职责范围，可通知项目组长或测试人员重新分配此 Bug。

4. 测试人员验证已修改的 Bug

开发人员处理完 Bug 并提交之后，测试人员查询开发者已修改的 Bug，即 Status 为“Resolved”，Resolution 为“Fixed”的 Bug，进行回归测试。经验证无误后，将缺陷的状态改为 Verified。待整个产品发布后，修改为 Closed。若还有问题，Reopened，状态重新变为“New”，并发邮件通知，如图 14.3 所示。

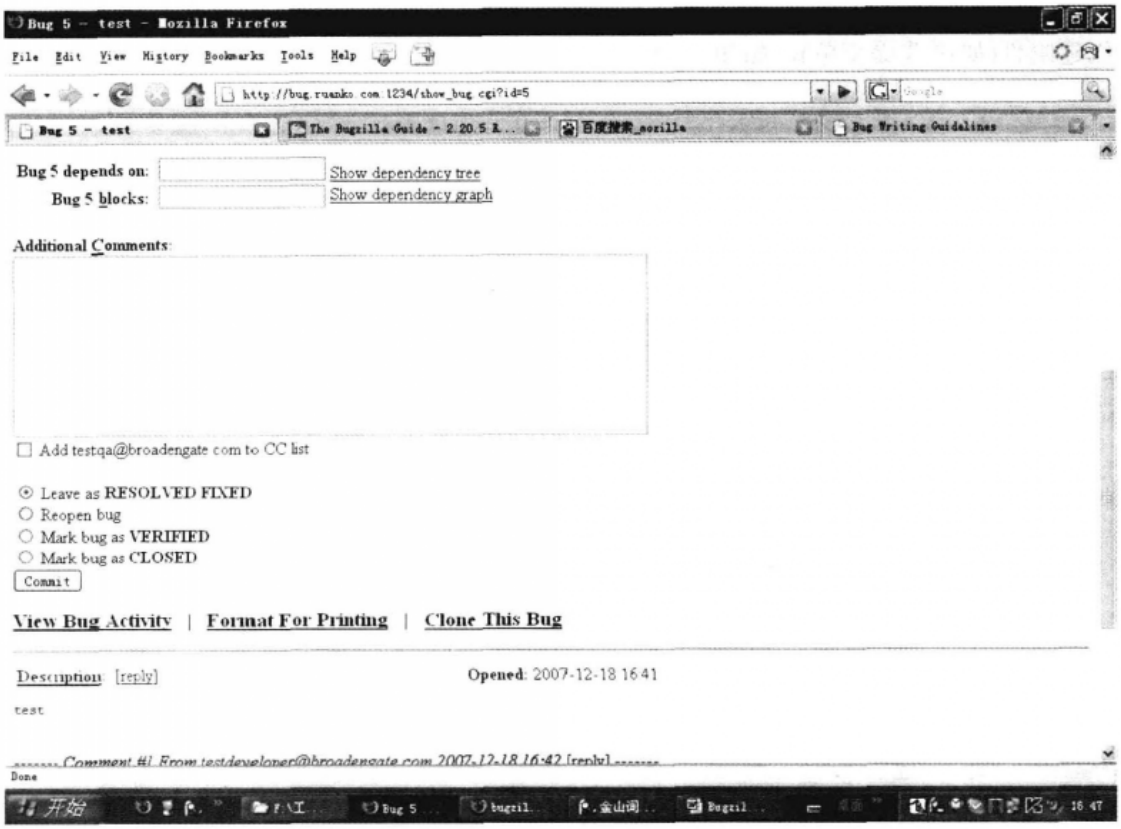


图 14.3 验证已修改的 Bug

5. 确认 Bug 是否存在

(1) 查询状态为“Unconfirmed”的 Bug。

(2) 测试人员对开发人员提交的 Bug 进行确认，确认 Bug 存在。

具体操作：选中“Confirm bug(change status to New)”后，进行 commit，操作结果为 Bug 的状态变为“New”。

6. 查找 Bug

可以使用 Bugzilla 的查询页面查找到系统中所有的 Bug 信息。Bug 报告中所有的字段信息都可作为查询条件。对于某些字段，可以选择多个值，在这种情况下，Bugzilla 会返回与任意值匹配的 Bug 记录。

可将一已执行的查询保存下来，成为一个“保存查询”(Saved Search)，显示在查询页面的页脚处，供以后重复使用。

可使用布尔表达式将多个查询条件组合在一起，构成高级查询。

7. 报表的生成

当使用查询功能得到 Bug 数据集后，可在此 Bug 数据集的基础上生成报表。Bugzilla 的报表分为两类：基于 HTML 表格的报表和基于图形的报表。其中基于图形的报表可绘制出线图、饼状图和柱状图。

例如，当使用查询功能查询出某一产品中的所有 Bug 记录后，可使用报表来显示出各

构件中不同严重程度的 Bug 的分布状况，从而发现哪些模块的质量存在严重问题。当定义好报表参数后，单击“Generate Report”，便可生成报表，且报表的形式可在 HTML 表格、线图、饼图和柱状图之间切换。

14.1.4　TestCenter 与 Testlink、Bugzilla 对比

Bugzilla 缺陷管理工具是一个错误跟踪系统，用于对软件产品程序开发过程的错误跟踪。作为一款开源的 Bug 追踪系统，是专门为 Unix 定制开发的。但是在 Windows 平台下依然可以成功安装使用。

Bugzilla 是一个拥有较强功能的错误跟踪系统，可以使开发测试人员在软件开发过程中跟踪软件错误的处理过程，为开发和测试工作以及产品质量的度量提供数据支持，从而保证软件产品的质量。

TestCenter 是一款更好更强大的缺陷管理、测试管理工具，拥有比 Bugzilla 更强大的功能以及无法比拟的优势。TestCenter 基于 B/S 体系结构，可以分别通过自动化测试或者手工测试制定测试流程，并且提供多任务的测试执行以及缺陷跟踪管理系统，最终生成测试报表。TestCenter 可以和本公司的测试工具 AutoRunner、Terminal AutoRunner 系列集成，也可以和其它测试工具集成，提供强大的自动化功能测试。

TestCenter 汇集了 QC、Bugzilla、bugfree 等众多优秀测试工具与缺陷管理工具的优点。下面从安装以及易用性上对比 TestCenter 与 Testlink、Bugzilla。

1. 安装方面

Testlink 和 Bugzilla 使用较为繁琐，修改配置文件比较麻烦，若要对这两款软件进行汉化的话，有可能出现乱码或数据丢失等现象。而 TestCenter 的整个安装过程只需 10 分钟即可，而且基本无需做什么配置，等待时间可以大大减少，提高了安装效率。

2. 功能方面

1) 缺陷管理

Testlink 无此功能。

Bugzilla 虽然有缺陷管理功能，但若汉化后会出现输入中文后变乱码、邮件无法发送等问题，而且 Bug 一旦变多，管理就会混乱，不能直接关联到相关的测试用例。

TestCenter 的缺陷管理考虑到中国用户的使用习惯，设置了多种查询功能，而且客户可以通过设置“过滤器”来查看满足自定义条件的 Bug。TestCenter 同样提供了导出 Buglist 的功能，而且支持打印。此外，在 TestCenter 的缺陷管理中可以查看所有 Bug 的图表分析，包括日分析和月分析，这大大提高了测试团队的管理效率。

2) 缺陷流程自定义

Testlink 无此功能，无从比较。

Bugzilla 的缺陷流程是比较规范的，但设计的时候并没有充分考虑到中小公司的使用。由于中小型公司的测试人员较少，势必会导致流程的不规范，这就关系到整个缺陷流程的自定义。在 Bugzilla 中提交 Bug 的整个流程是固定的，不可能根据中小型公司的实际需要来改变整个流程。

TestCenter 不仅满足一些大公司的测试需求，而且考虑到了中小型企业的实际情况，可

以在缺陷自定义流程中自己做一些修改，比方说每一步都有上传附件和修改备注信息等功能。也可由测试员自己关闭 Bug，在提交完毕后还可以查看每一步所做的具体修改信息。

3) 缺陷管理(关联测试用例)

Testlink 和 Bugzilla 都无此功能。

缺陷管理中的关联测试用例是 TestCenter 的一个特色。考虑到用户可能由于 Bug 的数量越多，管理越混乱，导致提交 Bug 后找不到关联的测试用例，因此 TestCenter 提供了关联测试用例功能。在提交 Bug 的过程中，用户可以填写与该 Bug 相关联的测试用例，到回归查看该 Bug 时就可方便地查询到关联的测试用例。

4) 缺陷管理(缺陷关联需求)

Testlink 和 Bugzilla 都无此功能。

一个完整的测试流程都会有明确的测试需求。所以，TestCenter 通过缺陷查看需求是十分重要的。在测试人员提交 Bug 的过程中，可以填写与该 Bug 相关的测试需求，到回归查看该 Bug 时就可方便地查询到关联的测试用例。

5) SQL 语句查询

Testlink 无此功能。

Bugzilla 虽然支持 SQL 语句的查询，但在查询过程中，一旦输错语句，则可能导致查询失败，而用户还未察觉到所输入的语句存在问题，进而无法在大量的缺陷中找到要查询的 Bug。

TestCenter 在最新版本中加入了 SQL 语句的查询功能，用户只需选择所要查询的条件，便可以自动生成 SQL 语句。若要查询其它缺陷，只需修改关键词即可，从而大大降低了输入 SQL 语句错误的概率。

6) 测试需求

Testlink 有新建测试需求的功能，但对于整个测试流程来说还是比较简单的，只能满足最基本的需要。

Bugzilla 无此功能。

TestCenter 有单独的测试需求模块，并且支持导出测试需求功能。TestCenter 的测试需求在 Testlink 的基础上还可以直接到测试用例模块中新建与测试需求完全相同的树结构，这就省去了新建完测试需求后还要到测试用例模块中建立对应的测试用例这一步骤，节省了大量宝贵的测试时间。不仅如此，TestCenter 的测试需求模块还提供了测试集新建向导，用户不必新建完测试需求后在每个模块中不断切换，便可直接在测试集模块中建立与需求相关的测试集。

7) 测试用例

Testlink 提供了测试用例，并且能够关联到测试需求，但每个测试用例并不能记录详细的操作过程，对于手工测试和自动化测试都不是很方便。

Bugzilla 无此功能。

TestCenter 的测试用例模块提供了导出测试用例组功能，并且在测试用例中可以设置组件(脚本)的依赖关系，在自动运行中可以方便地按照所要执行的顺序进行自动运行。此外，新版本的 TestCenter 提供了“用例设计”模块。在“用例设计”中可以详细记录自动执行或手动执行时每一步的详细步骤，这些在 Testlink 中都没有提供。

8) 手动测试任务管理

TestCenter 提供了面向任务的测试管理：第一，从测试计划发起一个测试活动为每一个测试用例创建一个测试任务；第二，测试经理可以把测试任务分配给测试工程师；第三，测试工程师根据测试任务来填报；第四，若测试过程中出现问题，可以直接根据测试问题提交缺陷。

9) 自动测试支持

TestCenter 提供了更多对自动测试的支持。自动测试是功能测试发展的方向，使用自动测试能够在更短的时间内执行更多轮次的测试。

TestCenter 的测试用例支持自动执行(通过调用业务组件的脚本)。Testlink 不支持自动测试。TestCenter 能够通过与自动测试工具连接，调用自动测试工具来执行测试脚本。

10) 对 BPT 的支持

TestCenter 支持面向业务流程的测试——Business Process Testing(BPT)。TestCenter 对 BPT 的支持，能够使自动测试类似于工作流等应用系统成为可能，提供更多的测试支持。

11) 支持业务组件

TestCenter 的测试对象支持三层组件：第一层，业务组件；第二层，测试用例；第三层，测试集。通过三层模型能够非常好地支持 BPT。

Testlink 只支持测试用例和测试集，无法提供对 BPT 的支持。

12) 从测试需求创建测试集

TestCenter 能够从需求出发来创建测试集，从而很好地计算测试需求的覆盖率。测试需求覆盖率对于测试经理而言非常重要：它是衡量测试覆盖范围的非常重要的因素。TestCenter 支持测试对比报表，能够对两次或者多次测试的结果进行比对，方便测试经理比对两次测试或者多次测试的结果。

14.2　问题跟踪软件——Jira

Jira 是澳大利亚 Atlassian 公司推出的一个问题跟踪管理软件，可跟踪和管理软件项目中出现的各种问题和缺陷。Jira 注重可配置性和灵活性，通过简洁易用的 Web 交互方式来满足技术用户和非技术用户的需求。目前 Jira 已经被 35 个国家的 2000 多个软件组织的项目管理人员、开发人员、分析人员、测试人员和其他人员所广泛使用。Jira 虽是商业软件，但对开源项目免费提供支持，因此在开源领域有很高的知名度。此外，在用户购买该软件的同时，可得到源代码，便于做二次开发。

14.2.1　Jira 的特点

Jira 具有以下特点：

(1) Jira 是“问题(issue)”管理工具，“问题”除包括缺陷外，还包括软件特征、任务和改进等。

(2) 具有高度可定制性，无论界面、缺陷跟踪工作流，还是问题属性字段，都可以由用户进行定制，使系统具有高度的灵活性和适应性。

(3) 能够跟踪软件组件和版本。

(4) 具有很强的查询功能，可执行全文搜索，并可将查询条件保存为过滤器，不同的用户可共享过滤器。

(5) 灵活的用户/用户组权限管理。可设置多个权限分配方案，不同的项目可以有不同的系统权限分配。

(6) 高度可配置的 E-mail 通知策略，不同的项目可设置不同的邮件通知方案。

(7) 具有对每个屏幕的打印选择。

(8) 具有对缺陷进行投票和监视的功能。

(9) 易于和其它系统(E-mail、SOAP、Excel、XML、CVS、SVN、Perforce 等)实现集成，监听器和服务程序能够提供与现有系统的双向信息交换。具有良好的可扩展性，完整的 JAVA 应用程序接口允许用户通过编写代码直接与 Jira 连接，从而可无限制地扩展 Jira。

(10) 具有很好的平台兼容性，能够运行于所有安装了 JDK 的操作系统上，并能够跟几乎所有的兼容 JDBC 的数据库一起使用。

14.2.2 缺陷跟踪操作

使用 Jira 可以很方便地进行缺陷跟踪流程中的各种操作。

1. 报告 Bug

先选择要报告的问题类型(Bug)，如图 14.4 所示。除 Bug 外，Jira 还可对其它各种类型的问题进行跟踪，如需求变更、软件改进等。

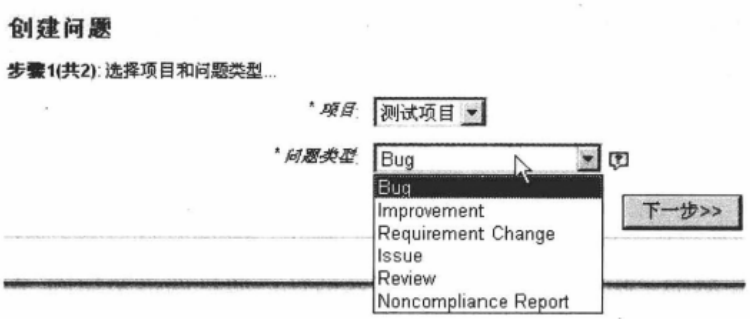


图 14.4 选择问题类型

单击“下一步”，出现 Bug 信息填写页面，如图 14.5 所示，在该页面中填写 Bug 的详细信息。

以下简要介绍图 14.5 中各信息字段的含义：

◆ 摘要：简明扼要地描述缺陷。

◆ 优先级：分为危急、严重、一般、次要、轻微等级别。

◆ 组件：选择缺陷所在的项目中的组件。

◆ 受影响版本：当前出问题的版本。

◆ 解决版本：规划在哪一个版本解决 Bug，一般为出问题的版本。

◆ 分配给：选择分配给特定的人员进行处理，如果不指定，则自动分配。

◆ 环境：例如操作系统、软件信息、硬件规格等信息。一般地，可在这里添加联系人、

联系方式等信息。

◆ 描述：对缺陷的详细描述。可附上出问题的 URL 地址，以方便追查故障。

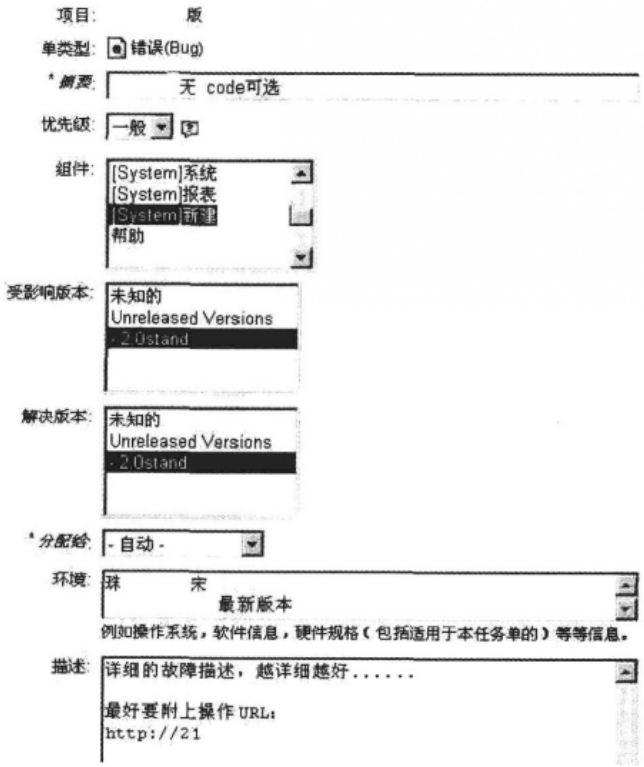


图 14.5　填写 Bug 的详细信息

2. 处理 Bug

开发人员首先查询分配给自己处理的缺陷，如图 14.6 所示。

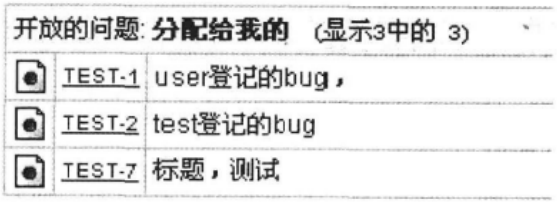


图 14.6　查询分配给自己处理的缺陷

接受缺陷处理任务，准备开始处理缺陷，如图 14.7 所示。

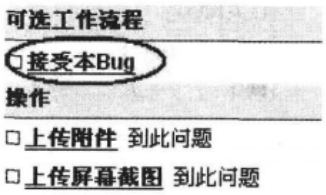


图 14.7　接受 Bug 处理任务

处理完缺陷后，准备填写处理情况，如图 14.8 所示。

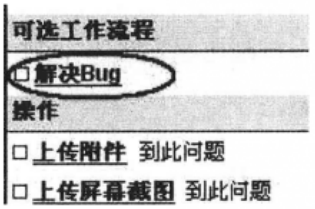


图 14.8 解决 Bug

填写缺陷处理情况，如图 14.9 所示。

图 14.9 填写缺陷处理情况

缺陷的处理方式有如下多种情况：

- Fixed：已修复。
- Later：在以后的版本中修复。
- Invalid：描述的问题不是一个 Bug。
- Won't Fix：该 Bug 将不会被修复。
- Duplicate：描述的问题与以前的某个 Bug 重复。
- Cannot Reproduce：不能重现该 Bug。

3. 关闭或重新打开 Bug

对于已经处理的缺陷，可将其关闭。若缺陷仍存在问题，可将其重新打开(对于已经关闭的缺陷，也可将其重新打开)，如图 14.10 所示。

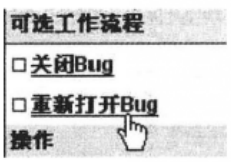


图 14.10 关闭或重新打开 Bug

14.2.3 查询操作

Jira 提供了很强的问题查询功能。在 Jira 的主菜单中单击“Find Issues”，可打开 Jira 的

问题浏览界面，在该界面左侧是问题查询表单，如图 14.11 所示。

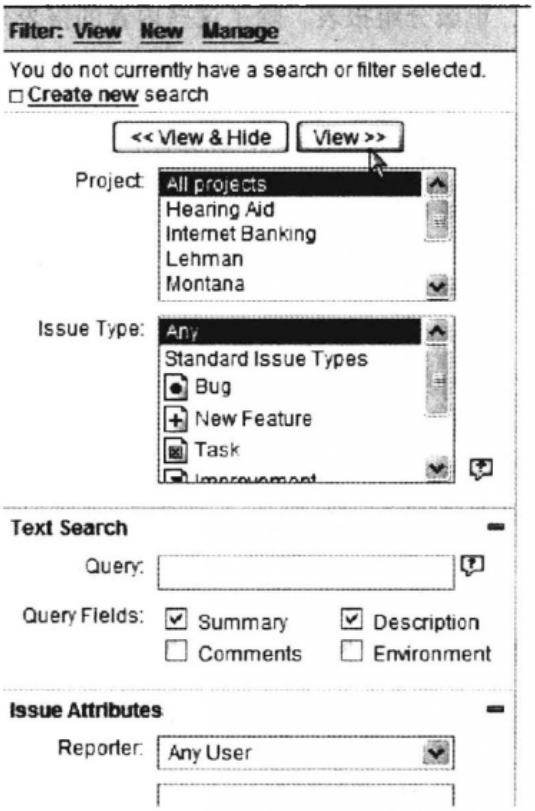


图 14.11　Jira 的问题查询界面

在图 14.11 所示的界面中可输入或选择各种查询条件，例如属于某项目的缺陷、描述字段包含了某一字符串的缺陷、某人员报告的所有缺陷，等等。查询条件输入完毕后，单击“View”按钮，即显示出所有满足条件的问题记录。

如果某一组查询条件可能会重复使用，可将其保存为“过滤器”。图 14.12 显示了 Jira 的过滤器界面。

在“过滤器”界面中，“View”选项卡用于查看当前的过滤器。该选项卡中显示了过滤器的名称、描述、摘要、可用操作等信息。一个过滤器可以被重命名、保存为新的过滤器、订阅或共享。“Edit”选项卡用于编辑当前过滤器的查询条件。“New”选项卡用于创建一个新的过滤器。“Manage”选项卡用于管理所有已存在的过滤器。

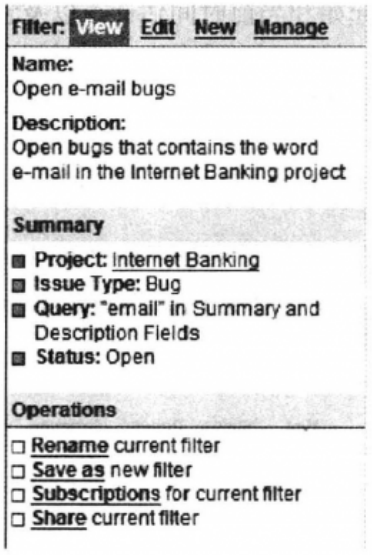


图 14.12　“过滤器”界面

14.2.4　生成报表

Jira 可产生各种各样的报表，这些报表显示了 Jira 系统中各项数据的统计结果。Jira 内

置的报表包括：路线图、变更日志、受关注问题、发布记录、时间跟踪报表、开发者工作量报表、版本工作量报表、单级分组报表。除了这些内置的报表外，还可通过插件的方式给 Jira 添加其它类型的报表。

1．路线图报表

Jira 为每个项目提供了一个路线图，显示出在将要发布的三个版本中需要解决的问题。路线图报表提供了对版本发布进展情况的概览。

单击 Jira 主菜单中的“Browse”，在项目列表中选择一个项目，然后单击“Road Map”选项卡，便可显示出该项目的路线图报表。

2．变更日志报表

变更日志报表显示了在某一项目最近的三个发布中所解决的问题，与路线图报表相反，变更日志报表提供了一个过去的视图，显示最近版本中已解决问题的概览。

单击 Jira 主菜单中的“Browse”，在项目列表中选择一个项目，然后单击“Change Log”选项卡，便可显示出该项目的变更日志报表。

3．受关注问题报表

受关注问题报表显示了一个项目中未被解决的问题，这些问题是按它们受关注的程度排序的，而受关注程度是按它们的“投票数”衡量的。因此该报表只有在“投票”功能有效时才能使用。

单击 Jira 主菜单中的“Browse”，在项目列表中选择一个项目，然后单击“Popular Issues”选项卡，便可显示出该项目的受关注问题报表。

4．时间跟踪报表

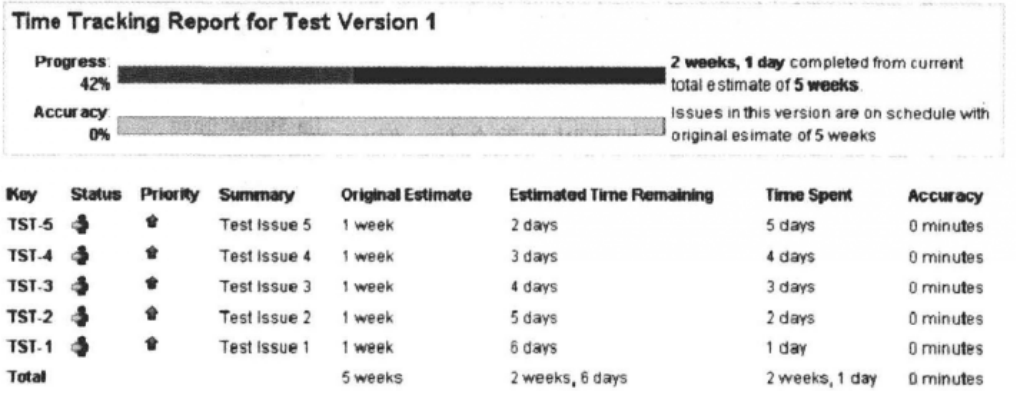
时间跟踪报表显示了一个特定产品版本中的问题的时间跟踪信息。它显示出特定问题的起始和当前时间估计，以及它们是否超前或滞后于起始的计划。

按以下步骤产生一个时间跟踪报表：

(1) 在项目浏览器页面中选择一个项目。

(2) 单击“Time Tracking Report”，打开时间跟踪报表参数设置表单。

(3) 填写必要的参数后，单击“report”按钮，生成类似图 14.13 所示的时间跟踪报表。



Key	Status	Priority	Summary	Original Estimate	Estimated Time Remaining	Time Spent	Accuracy
TST-5			Test Issue 5	1 week	2 days	5 days	0 minutes
TST-4			Test Issue 4	1 week	3 days	4 days	0 minutes
TST-3			Test Issue 3	1 week	4 days	3 days	0 minutes
TST-2			Test Issue 2	1 week	5 days	2 days	0 minutes
TST-1			Test Issue 1	1 week	6 days	1 day	0 minutes
Total				5 weeks	2 weeks, 6 days	2 weeks, 1 day	0 minutes

图 14.13　时间跟踪报表

5. 开发者工作量报表

开发者工作量报表显示了分配给某一特定用户的问题的时间跟踪信息，它显示出未解决的分配给用户的问题数量和剩余的工作量。

按照如下步骤产生开发者工作量报表：

(1) 在项目浏览器页面中选择一个项目。

(2) 单击“Developer Workload Report”。

(3) 选择一个用户，单击“Next”，生成报表。

6. 版本工作量报表

版本工作量报表显示了某项目中特定产品版本的当前工作量的时间跟踪信息。对于一个特定版本，该报表显示出未解决的分配给每个用户的问题数量和剩余工作量。

产生版本工作量报表的步骤如下：

(1) 在项目浏览器页面中选择一个项目。

(2) 单击“Version Workload Report”。

(3) 选择一个产品版本，单击“Next”，生成报表。

7. 单级分组报表

单级分组报表显示出某一过滤器所查询到的所有问题，这些问题按照一个特定的字段分组。例如：一个过滤器用于查询某一特定版本中所有打开的 Bug，则可设计一单级分组报表，按照 Bug 处理人员分组显示这些 Bug。

产生版本工作量报表的步骤如下：

(1) 在项目浏览器页面中选择一个项目。

(2) 单击“Single Level Group by Report”。

(3) 选择过滤器和分组字段，单击“Next”，生成如图 14.14 所示的报表。

图 14.14　单级分组报表

14.2.5 系统设置

Jira 的一大特点就是非常灵活，它提供了各种系统设置功能，从而可以适应不同的应用环境。Jira 可对以下各项提供定制功能：

- ◆ 问题信息字段：可灵活地定制描述问题的信息字段。
- ◆ 工作流：可定制问题跟踪工作流。
- ◆ 权限分配方案：灵活地为用户组和用户分配操作权限。
- ◆ 邮件通知方案：定义当问题状态发生改变时，应邮件通知哪些用户/用户组。
- ◆ 操作界面：可自定义操作界面上的信息项和信息项的分布。

第 15 章　单元测试工具

本章详细介绍单元测试工具 Junit。对 Junit 的下载安装、特点、使用方法等进行了说明。

15.1　Junit 的安装

首先给出 Junit 3.8.1 版本的安装步骤，如下所示：

(1) 下载 Junit。登录 Junit 的网站(http://www.junit.org/)，从该网站可以下载 Junit，并可找到相关资料。

(2) 解包 Junit，如表 15.1 所示。

表 15.1　Junit 包含的文件及其目录

文件/目录	描　述
Junit.jar	Junit 框架结构、扩展和测试运行器的二进制发布
Src.jar	Junit 的源代码，包括一个 Ant 的 Buildfile 文件
Junit	一个目录，内有 Junit 自带的用 Junit 编写的测试示例程序
Javadoc	Junit 完整的 API 文档
doc	一些文档，包括 TestInfected：Programmers　Love Writing Tests 等，帮助用户入门

(3) 检验安装 Junit。检验 Junit 是否安装正确，执行 Junit 自带的测试示例程序，详细步骤如下所示：

① 打开命令行提示窗口。

② 将 Junit 的目录(Windows 系统下 C：\junit3.8.1 或 linux 系统下 opt\junit3.8.1)作为当前目录。

③ 执行下列命令：

```
>java –classpath junit.jar;.junit.textui.TestRunner junit.samples.AllTests
```

执行测试命令，类路径包含了 junit.jar 和当前的目录(.)。junit.jar 是仅有的需要放到类路径下的文件。当前的目录(.)是解包 Junit 的目录，Junit 测试的所有*.class 文件从此开始。junit.textui.TestRunner 是 Junit 的基于文本的测试运行器的类名，会执行所有的 Junit 测试，并将结果报告给控制台。junit.samples.AllTests 是运行测试套件的名字。

(4) 运行 Junit 测试。

15.2　Junit 的特点

Junit 用于单元级测试的开放式框架，具有如下优势：

(1) Junit 是完全免费的。Junit 是公开源代码的，可以进行二次开发。

(2) 使用方便。Junit 可以快速地撰写测试并检测程序代码，随着程序代码增加测试用例，Junit 执行测试类似编译程序代码一样容易。

(3) Junit 检验结果并提供立即回馈。Junit 自动执行并且检查结果，执行测试后获得简单回馈，不需要人工检查测试结果报告。

(4) Junit 合成测试系列的层级架构。Junit 把测试组织成测试系列，允许组合多个测试并自动回归整个测试系列。Junit 与 Ant 结合实施增量开发和自动化测试。

(5) Junit 提升软件的稳定性。Junit 使用小版本发布，控制代码更改量。同时，引入了重构概念，提高软件代码质量。

(6) 与 IDE 的集成。与 Java 相关的 IDE 环境集成，实现测试及开发代码之间无缝连接。

15.3 Junit 的内容

Junit 作为单元测试框架，共有六个包，其中最核心的三个包是 Junit.framework、Junit.runner 和 Junit.textui。Junit.framework 是测试构架，包含了 Junit 测试类所需的所有基类；Junit.runner 负责测试驱动的全过程；Junit.textui 负责文字方式的用户交互。

(1) Junit.framework 共有 6 个主要类或接口，分别是 Test、Assert、TestCase、TestSuite、TestListener 和 TestResult，如下所示：

① Test 接口。Test 接口作为单独测试用例(TestCase)、聚合测试用例(TestSuite)的共同接口，从而符合该接口的所有测试形态都被 Junit 使用一致的方式处理，同时为整个框架做了扩展预留。

② Assert 类。Assert 类包含一组用于测试断言方法的集合，验证期望值与实际值是否一致。如果期望值和实际值比对失败，Assert 类就会抛出个 AssertionFailecdError 异常，Junit 测试框架将这种错误归入 Failes 并加以记录。

③ TestCase 抽象类。TestCase 抽象类继承 Assert 类，实现 Test 接口，负责进行初始化以及测试方法调用。TestCase 类作为包的核心，是抽象类，只能通过实例化对象实现。

TestCase 类包括 setUp()、tearDown()方法。setUp()方法测试所有变量和实例，并且在依次调用测试类中每个测试方法之前再次执行 setUp()方法，使得每次所生成的变量、实例和第一次初始化时相同。tearDown()方法则是在每个测试方法执行后，准确释放测试方法中引用的变量和实例。

④ TestSuite 类。TestSuite 类实现了 Test 接口，可以包装、组织和运行多个 TestCase。

TestSuite 处理 TestCase 有 6 个规约，否则便会拒绝执行测试。这 6 个规约如下所示：

- 该测试用例必须是公有类。
- 该测试用例必须继承于 TestCase 类。
- 测试用例中测试方法必须是公有的(Public)。
- 测试用例中测试方法必须被声明为 Void。
- 测试用例中的测试方法的前置名词必须是 test。
- 测例中测试方法无任何传参。

TestSuite 处理的测试用例标准写法：

```
//必须声明为 Public 类，继承于 Junit.framework.TestCase 类
Public class Class_TestCase extends TestCase{
//标准测试用例构造方法无需变动
Public Class_TestCase()  {  //必须声明为 public
Super();                    //默认写法一般不用重写
}
Public   void testAMethod(){ …}  //测试方法必须声明为 Public，并且加上 “test”
                                 //修饰前缀
Public   void testBMethod(){ …}
}
```

⑤ TestListener。TestListener 作为一个事件监听规约，定义了执行测试过程的公共方法，通知 Listener 的对象相关事件，包括测试开始、错误和失败的抛出、测试结束，如 startTest()、endTest()等。

⑥ TestResult 类。TestResult 类负责收集 TestCase 所执行的结果，将结果分为两类，即客户可预测的失败(Failure)和没有预测的错误(Error)。Failure 表示当 Junit 测试结果为 False 时，TestResult 会自动抛出 AssertionFailedErrors 异常。Error 表示测试驱动程序本身的错误，作为不可预见的异常情况，由测试代码自身抛出。TestResult 提供 wasSuccessful()方法，决定所做的测试是否全部通过。

TestResult 对 TestListener 进行注册(每个测试用例对应一个异常监听者)，TestListener 调用测试方法后向 TestResult 返回测试执行过程，例如测试的整个执行生命周期包括测试开始、失败和错误的抛出、测试结束。

(2) Junit.runner 包中定义 Junit 测试框架的交互形式，也是整个 Junit 的交互框架。BaseTestRunner 抽象类是 Junit.runner 包的核心类，用于实现 TestListener 接口，定义运行测试的公共方法。所有 Junit 框架和外界进行交互的行为都被此包所定义。BaseTestRunner 抽象类分别被 Junit 中 awtui、swingui 和 textui 三个包中同名的 TestRunner 方法共同继承，形成 3 种不同风格的 Junit 交互模式。

一般来说，命令行交互模式执行测试速度最快，界面简单，返回的错误值集成到 Ant 中进行后续处理。

图形交互模式执行测试，采用 3 种色块：灰色、绿色、红色标注测试分组，给出相关测试失败的错误原因。其中，灰色代表羞涩，表示单元代码的错误输出；绿色等同于活跃的生命，表示结果正确；红色表示当前代码出现了严重的错误。

(3) Junit.textui 包中主要的类是 TestRunner，继承了 BaseTestRunner，是客户对象调用的起点，负责对整个测试流程跟踪，显示返回测试结果，报告测试进度。

15.4　Junit 的设计原则

Junit 不仅是验证程序的正确性或者是一种发现 Bug 的工具，而且是为了验证被测试代码是否实现了符合预期设计而存在。Junit 重构功能如下所示。

步骤一：编写单元级测试代码，进行测试用例的设计。

步骤二：编写代码通过单元级测试。

步骤三：重构的运用。

步骤四：重新运行测试。

下面给出 Junit 测试的设计原则。

1) 不要测试简单的情况

Junit 只是一个优秀的单元级测试架构，并没有规定要测试些什么。一般来说，被测试类的每一个公共方法对应一个测试方法，但是对于一些不可能出错的方法，例如 Set 和 Get 方法，这样的做法就没有任何意义了。

2) 测试任何可能出现错误的地方

极限编程(XP)的测试原则之一是不放过任何可能出错的地方。如果类复杂，则完全测试的难度较大；反之，如果类简单，则完全测试的可能性就大。Junit 支持重构，强调类在功能上尽可能简单易理解。

3) 测试边界条件

边界条件必须考虑可能的溢出，例如集合是否为空、系统内存地址的溢出、数组的第一个和最后一个元素。

通常需要考虑的边界条件有：

(1) 未初始化：很多编译器能够在某种情况下给出对象没有初始化的信息提示，但是更多的隐藏未初始化情况被忽略。

(2) Null 值：如果输入 Null 值，代码该如何处理，是否会抛出指定的异常情况。

(3) 最大值、最小值：第一个和最后一个是必然的选择。

(4) 临界值：如果超过最大值或者小于最小值，是否会抛出指定的异常情况。

(5) 初始值：不同条件语句的初始值不同，是 0 还是 1 或者循环次数是 i++还是--i 等。

4) 自动化

Junit 单元级测试必须被自动化，对于重构代码的更新意味着能快速反馈。另外，自动化测试也意味着对测试结果自动评价其是否符合预期值的设定。

5) 测试依赖于接口

利用类接口进行测试是一种策略，即测试要依赖于对象接口的实现。从设计上来看，频繁地测试一个类的非接口方法是不正常的，这意味着过多地依赖于类的实现而非类的接口。

15.5 测 试 示 例

通过以下几个步骤，Junit 完成简单的测试：

(1) 创建 TestCase 类的一个子类。

(2) 编写若干测试用例，每个测试用例书写格式如下所示：

```
Public void test<TestCaseName>(){ … }
```

注意：Junit 对于测试用例的命名法是 test+<TestCaseName>测试用例的名字。

(3) 编写一个测试套件方法加入第(2)步编写的测试用例。

```
Public static Testsuite(){…}
```

编译上述子类以及被测构件，用 Junit 提供的运行器 TestRunner 运行测试。

【例 15-1】　Junit 测试的简单范例。

步骤如下：

步骤 1：创建一个 TestCase 的子类。

```
package junitfaq;
import java.util.*;
import junit.framework.*;

public class SimpleTest extends TestCase {

public SimpleTest(String name) {
super(name);
}
```

步骤 2：写一个测试方法断言期望的结果。

```
public void testEmptyCollection() {
Collection collection = new ArrayList();
assertTrue(collection.isEmpty());
}
```

注意：Junit 推荐的做法是以 test 作为待测试的方法的开头，这些方法可以被自动找到并被测试。

步骤 3：写一个 suite()方法，使用反射动态地创建包含 testXxxx 方法的测试套件。

```
public static Testsuite() {
return new TestSuite(SimpleTest.class);
}
```

步骤 4：运行测试。

方法一：文本方式。

在 main()方法里调用 junit.textui.TestRunner.run(…)，具体代码如下所示：

```
public static void main(String args[]) {
junit.textui.TestRunner.run(suite());
}
}
```

运行结果如图 15.1 所示。

图 15.1　【例 15-1】文本方式运行结果

分析测试如下所示：Time 上的小点表示测试个数，如果测试通过则显示 OK；否则在小点的后边标上 F，表示该测试失败。

Junit 报告结果为 OK，表明测试成功；反之，根据 Junit 提示的错误信息进行修正。

方法二：图形方式。

采用如下语句，其执行结果如图 15.2 所示。

```
java junit.swingui.TestRunner junitfaq.SimpleTest
```

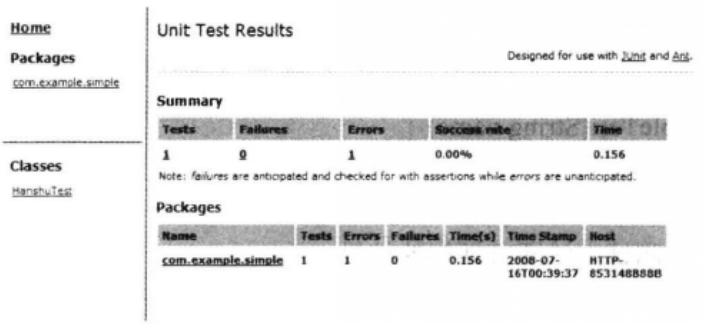


图 15.2 【例 15-1】图形方式运行结果

实际测试某个类功能常常需要执行一些共同的操作，完成以后需要销毁所占用的资源(例如网络连接、数据库连接，关闭打开的文件等)。TestCase 类提供的 setUp 方法在每个 testXxxx 方法之前运行，而 tearDown 方法在每个 testXxxx 方法结束以后执行，既共享了初始化代码，又消除了测试代码之间的相互影响。

【例 15-2】 判断三角形。

(1) Triangle 类作为三角形类，用于求解三角形的周长、面积等，代码如下所示。

```
import java.lang.Math;
public class Triangle
{
    int a，b，c;
    double area，len;
    public void set(int i，int j，int k)
    {
        a=i;
        b=j;
        c=k;
    }
    public boolean Judge(int a，int b，int c)
    {
        boolean bool;
        if(a+b>c && a+c>b && b+c>a)
        {
            bool=true;
            return bool;
```

```
            }
            else
                return bool=false;
        }
        public int GetPerimeter(int a，int b，int c)
        {
            return (a+b+c)/2;
        }
        public double GetArea(int a，int b，int c)
        {
            len=(a+b+c)/2;
            area=Math.sqrt(len*(len-a)*(len-b)*(len-c));
            return area;
        }
    }
```

(2) 设计 TriangleTest 类实现对 Triangle 类进行测试，代码如下所示。

```
    import junit.framework.*;
    import org.junit.Assert.*;
    import java.util.*;

    public class TriangleTest extends TestCase {
          Triangle testTriangle=new Triangle();      //初始化类对象

        public void setUp() throws Exception {
            super.setUp();
        }
        protected void tearDown() throws Exception {
            super.tearDown();
        }
        public void testSet(int a，int b，int c) {      //测试边长 a，b，c 的值
            assertNull(a);
            assertNull(b);
            assertNull(c);
        }

        public void testJudge() {                       //测试是否符合三角形的定义
            assertTrue(testTriangle.Judge(3, 4, 5));
        }
```

```
    public void testGetPerimeter() {                    //测试三角形周长
        assertEquals(6，testTriangle.GetPerimeter(3, 4, 5));
    }

public void testGetArea() {                             //测试三角形面积
        assertEquals(6，testTriangle.GetArea(3, 4, 5));
    }
    public static Testsuite() {
return new TestSuite(TriangleTest.class);
}
```

(3) 运行测试。

方法一：文本方式。

```
    public static void main(String[] args){
    junit.textui.TestRunner.run(TriangleTest.suite());
}
```

结果若是错误，则会有相应的提示，如图 15.3 所示。

图 15.3 【例 15-2】文本方式运行结果

方法二：图形方式，运行结果如图 15.4 所示。

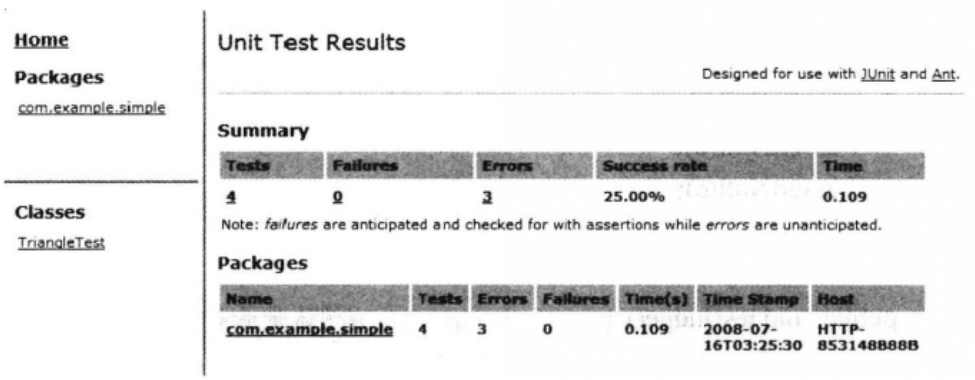


图 15.4 【例 15-2】图形方式运行结果

第 16 章　功能测试工具

功能测试工具的一般原理是利用脚本的录制(Record)/回放(Playback)，模拟用户的操作，然后将被测系统的输出记录下来同预先给定的标准结果比较。黑盒测试工具的代表有：Rational 公司的 TeamTest、Robot；Compuware 公司的 QACenter。本章介绍 MI 公司的功能测试工具 WinRunner 和 Quick Test Professional。

16.1　WinRunner 简介

WinRunner 最主要的功能是自动重复执行某一固定的测试过程，它以脚本的形式记录下手动测试的一系列操作，在环境相同的情况下重放，检查其在相同的环境中有无异常的现象或与实际结果不符的地方。WinRunner 可以减少由于人为因素造成的结果错误，同时也可以节省测试人员的大量时间和精力。

在测试执行过程中，通过 WinRunner 记录所有必需的操作，设置检查点，考察应用程序的各个对象，把过程保存为脚本。应用程序更新后，通过对脚本的运行，重复先前的操作，可检查错误是否已被修改、是否有新错误被引入等。

本节介绍 WinRunner 8.02，其内容主要包括 WinRunner 的两种测试模式、测试过程，并通过两个实例说明了 WinRunner 的使用方法。

16.1.1　WinRunner 测试模式

当用户在软件操作中点击 GUI(图形用户界面)对象时，WinRunner 会用一种类 C 的测试脚本语言(TSL)生成一个测试脚本。用户可以根据需要用手工编程的方法编辑这个脚本。利用 WinRunner 的功能生成器(Function Generator)可以快速简便地在已录制的测试中添加功能。

WinRunner 包括两种录制测试的模式。

1．环境判断模式(Context Sensitive mode)

环境判断模式根据用户选取的 GUI 对象(如窗体、清单、按钮等)把对软件的操作动作录制下来，并忽略这些对象在屏幕上的物理位置。每一次对被测软件进行操作，测试脚本中的脚本语言都会描述选取的对象和操作动作。

进行录制时，WinRunner 会对用户选取的每个对象做唯一描述并写入 GUI map(映射)中。GUI map 和测试脚本被分开保存维护。当软件用户界面发生变化时，只需更新 GUI map。这样一来，环境感应模式的测试脚本将非常容易地被重复使用。

执行测试只需要回放测试脚本。WinRunner 模拟一个用户使用鼠标选取对象、用键盘输

入数据。WinRunner 从 GUI map 中读取对象描述，并在被测软件中查找符合这些描述的对象。WinRunner 可以在同一个窗体中找到这些对象。

2．模拟模式(Analog mode)

模拟模式记录鼠标点击、键盘输入和鼠标在二维平面上(x 轴和 y 轴)的精确运动轨迹。执行测试时，WinRunner 让鼠标根据轨迹运动。这种模式对于那些需要追踪鼠标运动的测试非常有用，例如画图软件。

16.1.2 WinRunner 测试过程

WinRunner 的测试过程可分六个步骤：

(1) 创建 GUI map。

(2) 创建测试。

(3) 调试测试。

(4) 执行测试。

(5) 查看测试结果。

(6) 报告发现的错误。

1．创建 GUI map

使用 RapidTest Script Wizard(快速测试脚本向导)回顾软件用户界面，并系统地把每个 GUI 对象的描述添加到 GUI map 中。也可以在录制测试的时候，通过点击对象把对单个对象的描述添加到 GUI map 中。

2．创建测试

用户可以通过录制、编程或两者同用的方式创建测试脚本。录制测试时，在需要检查软件反应的地方插入检查点(Checkpoint)。可以插入检查点来检查 GUI 对象、位图(Bitmap)和数据库。在这个过程中，WinRunner 捕捉数据，并作为期望结果(被测软件的期望反应)储存下来。

3．调试测试

用户可以先在调试模式(Debug mode)下运行脚本，也可以设置中断点(Breakpoint)，监测变量，控制 WinRunner 识别和隔离错误。调试结果被保存在 Debug folder 中，一旦调试结束就可以删除。

4．执行测试

检验模式(Verify mode)下测试被测软件。WinRunner 在脚本运行中遇到检查点后，就把当前数据和前期捕捉的期望值进行比较。如果发现有不符合，就记录下来作为实测结果。

5．查看测试结果

测试是成功还是失败由用户来认定。每次测试结束，WinRunner 会把结果显示在报告中。报告会详述测试执行过程中发生的所有主要事件，如检查点、错误信息、系统信息或用户信息。

如果在检查点处有预期结果与实测结果不符合的情况，可以在 Test Results(测试结果)窗口查看预期结果和实测结果。如果是位图不符合，也可以查看用于显示预期值和实测结

果之间差异的位图。

6．报告发现的错误

如果由于测试中发现错误而造成测试运行失败，用户可以直接从 Test Results 窗口报告有关错误的信息。这些信息通过 E-mail 发送给测试经理(QA Manager)，用来跟踪这个错误直到被修复。

16.1.3　认识 WinRunner 工作环境

1．启动 WinRunner

点击“开始”|“程序”|“WinRunner”|“WinRunner”，启动 WinRunner，WinRunner 的 Record/Run Engine(记录/执行引擎)图标即出现在 Windows 的任务条上。这个引擎设立和维护 WinRunner 与被测软件之间的连接。

在 WinRunner 启动时，可以选择支持 ActiveX control、PowerBuilder、VisualBasic 或 WebTest 的插件。其它插件需要单独再向 MI 公司购买，建议不要同时载入所有的插件，不必要的插件可能会对录制或执行造成问题，如图 16.1 所示。

图 16.1　启动插件选择

2．WinRunner 主界面介绍

1) WinRunner 主窗口

WinRunner 主窗口包括以下几部分，如图 16.2 所示。

(1) WinRunner title bar：标题栏；

(2) Menu bar：菜单栏；

(3) Standard toolbar：标准工具栏，包含运行测试时常用的命令；

(4) Test Editor：脚本编辑栏；

(5) User toolbar：用户工具栏，包含创建测试时常用的命令；

(6) Status bar：状态栏；

(7) Function Viewer：功能窗口；

(8) Debug Viewer：调试窗口。

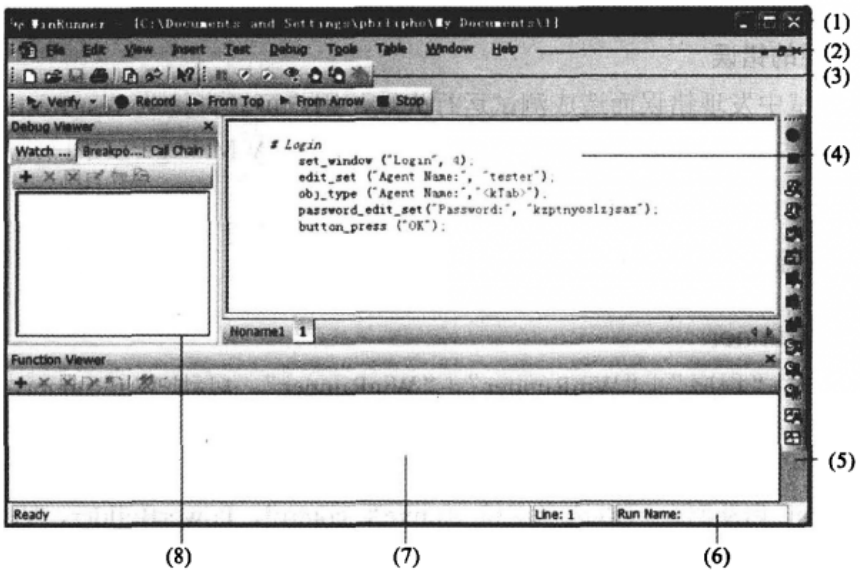


图 16.2 WinRunner 主窗口

2) 测试窗口

在测试窗口创建和执行测试，窗口包含以下部分，如图 16.3 所示。

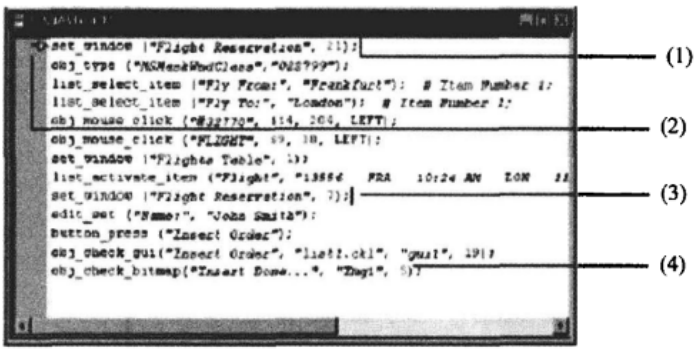


图 16.3 WinRunner 测试窗口

(1) Test script：测试脚本，通过录制或编写代码方式生成；

(2) Execution arrow：执行箭头，指明当前正在执行的那一行脚本，如果想要移动这个标志到某一行，只需要在执行改行左侧空白处单击鼠标右键；

(3) Insertion point：插入点，用户可插入或编辑文本的地方；

(4) Test scrip：测试脚本。

3) 测试工具列

提供常用的测试按钮，如录制、停止等，如图 16.4 所示。

(1) Run Mode：设定执行模式，有 Verifty、Debug、Update 三种执行模式；

(2) Record：开始以 Context sensitive 模式录制；

(3) Run From Top：从头开始执行；

(4) Run From Arrow：从黄色小箭头处开始执行；

(5) Stop：停止录制或执行。

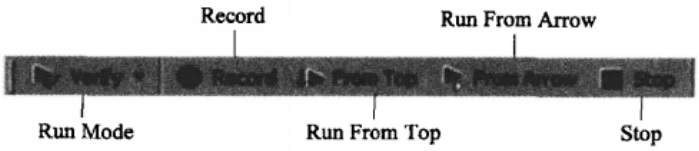


图 16.4　WinRunner 测试工具列

4) 除错工具列

提供除错时常用的测试按钮，如逐步执行、暂停、设定断点等，如图 16.5 所示。

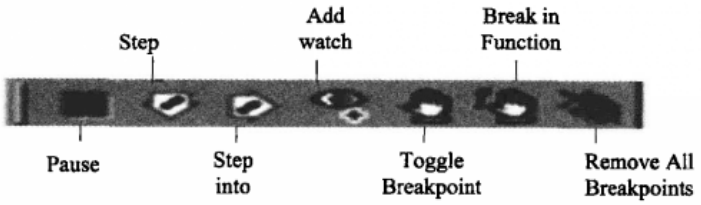


图 16.5　WinRunner 除错工具列

(1) Pause：暂停；

(2) Step：逐步执行；

(3) Step Into：逐步执行并进入；

(4) Add watch：新增监视变量；

(5) Toggle Breakpoint(F9)：设置断点；

(6) Break in Function(Ctrl+B)：设定函数中的断点；

(7) Remove All Breakpoints：清除所有断点。

5) 用户工具列

提供使用者自定义常用的按钮，如图 16.6 所示。

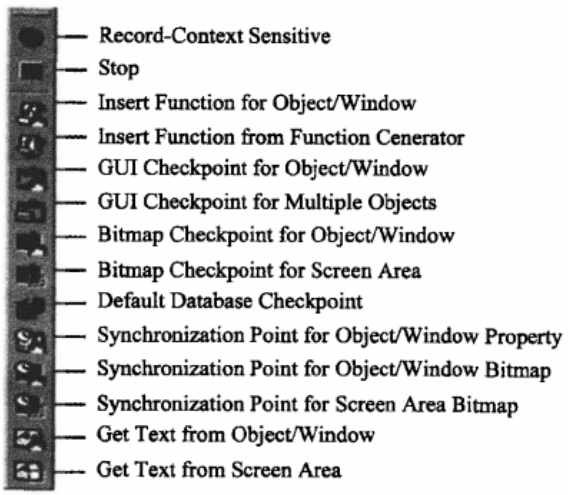


图 16.6　WinRunner 用户工具列

16.1.4　WinRunner 测试示例一

下面通过 WinRunner 自带的飞机订票的例子，来讲述如何利用 WinRunner 进行自动化

功能测试。通过举例可以看到，测试过程中的一个或多个步骤可以根据实际需要省略。

1. 创建 GUI map

当创建测试时，用户需要确定 GUI map 的工作模式。GUI map 有两种工作模式：Global GUI Map file 和 GUI Map file per test。

对于 WinRunner 初学者，建议使用 GUI Map file per test 模式。这种模式下，每次新建测试就自动新建一个 GUI map file。在用户保存测试时，GUI map file 自动保存；用户打开测试时，GUI map file 自动加载。

对于 WinRunner 熟练的开发者，建议使用更有效率的 Global GUI Map file 模式，这也是 WinRunner 的默认模式。WinRunner 8.02 或更低版本都是使用这种模式，且只能使用这种模式。

本书中使用 GUI Map file per test 模式。

更改 GUI map 的工作模式：在 WinRunner 主菜单栏中选取“Tools”|“General Options”|“GUI Files”，可以选择使用 GUI Map file per test 模式，如图 16.7 所示。注意，WinRunner 重新启动后设置才会生效。

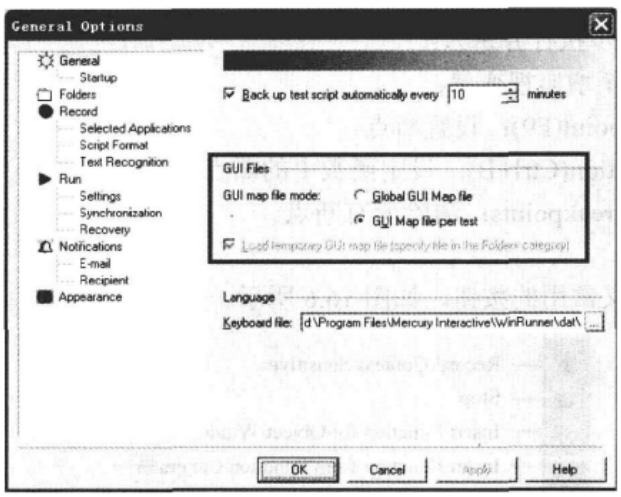


图 16.7 WinRunner 更改 GUI map 工作模式界面

2. 创建测试

(1) 运行 WinRunner，新建一个测试项目：在 WinRunner 主菜单栏中点击“File”|“New”。

(2) 运行 Flight 4A 程序，进入“Login”窗口：点击“开始”按钮，选择“开始”|“程序”|“WinRunner”|“Sample Applications”，点击“Flight 4A”快捷方式，显示出示例程序的登陆界面，如图 16.8 所示。

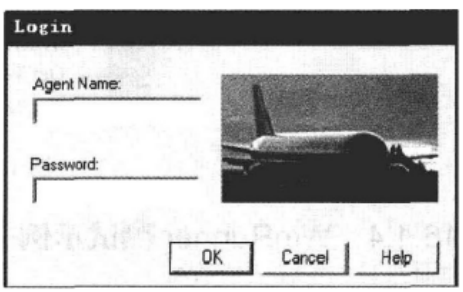


图 16.8 示例程序 Flight 4A 登录界面

(3) 开始录制：在 WinRunner 工具栏点击“Record”按钮，WinRunner 进入录制状态。

(4) 执行登录程序：在 Flight 4A 示例程序登录界面，点击“Agent Name”输入框，输入“admin”；点击“Password”输入框，输入“mercury”；点击“OK”

按钮，程序登录成功，进入主界面。

(5) 停止录制：在 WinRunner 主菜单栏点击“Stop”按钮，可以看到 WinRunner 中记录的脚本如下：

```
# Login                                                    1
set_window ("Login",  4);                                  2
edit_set ("Agent Name:",  "admin");                        3
obj_type ("Agent Name:", "<kTab>");                        4
password_edit_set("Password:",  "kzptnyoslzjsaz");         5
button_press ("OK");                                       6
```

第一行：注释。

第二行：获取登录“Login”对话框。

第三行：在“Agent Name:”编辑框中输入 admin。

第四行：用户按了 Tab 键。

第五行：在“Password:”编辑框中输入密码 mercury，脚本显示为加密后的文本。

第六行：用户按了“OK”按钮。

(6) 保存脚本：保存脚本为“C:\TestProject\WR_Lesson_Record_and_Run”。GUI Map 可以自动保存在用户目录下，也可选择手动保存。本例中用户无需手动保存。手动保存 GUI Map 到“C:\TestProject\WR_Lesson_Record_and_Run\flight1a.gui”，方法是：选择菜单“Tools”|“GUI Map Editor”，显示“GUI Map Editor”窗口，如图 16.9 所示；选择“GUI Map Editor”窗口中的菜单“File”|“Save”，输入要保存的路径和文件名，点击“保存”按钮即可。建议：将 GUI Map 文件保存到当前脚本的目录中。

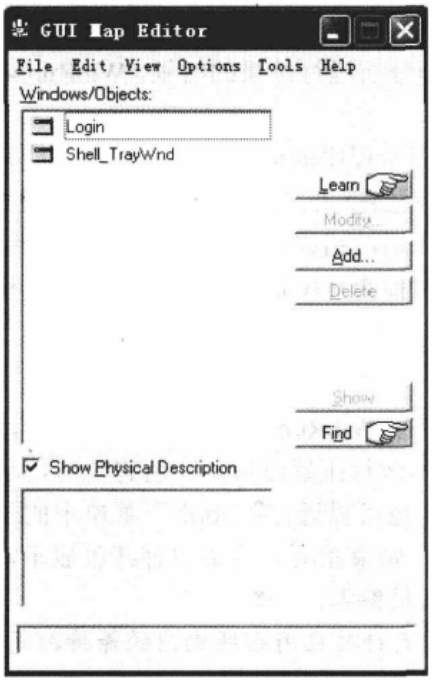


图 16.9　WinRunner GUI Map 文件保存界面

3. 修改测试脚本

录制一个脚本的时候，每次点击 GUI 对象或用键盘输入时，在 WinRunner 的测试脚本中产生了一个 TSL 的声明。可以增加录制 TSL 的函数，TSL 包括许多构造函数，能增加脚本测试函数的功能和适应性。在 WinRunner 中通过形象规划工具函数发生器能很快地增加测试脚本的功能。测试人员可以通过函数发生器为脚本定义函数或者通过其选择相应的函数。

在刚才录制的脚本中，增加一段程序 wait(2)，这是 WinRunner 的一个函数，功能为设定等待时间，时间一到就执行后面的脚本。合理使用不同的函数可以增强脚本的可用性。

```
# Login
set_window ("Login", 4);
edit_set ("Agent Name:", "admin");
obj_type ("Agent Name:", "<kTab>");
wait(2);                                                    //增加一行
password_edit_set("Password:", "kzptnyoslzjsaz");
button_press ("OK");
```

4. 执行测试

(1) 关闭 Flight 4A 程序登录后进入的“Flight Reservation”窗口。

(2) 重新运行 Flight 4A 程序，等待“Login”窗口的出现(如果计算机能够流畅地运行 WinRunner 的话，“Login”窗口应该非常快地出现)。

(3) 回放刚才录制的脚本：在 WinRunner 主菜单栏中点击“Run from Top”，可以看到刚才对“Login”窗口所做的操作和输入均被重现了一遍。由于加入了 wait(2)函数，因此输入用户名后，等待 2 秒后才输入密码。其它效果和手工进行的操作与输入的结果一样。

执行所记录的测试脚本和分析运行结果的时候，WinRunner 提供了三种运行模式，可以从工具栏中选择运行模式。

◆ 运行测试脚本，检测应用程序的运行情况，察看各检查点内容是否和预期一致，并保存测试结果，使用 Verify(校验)模式。该模式为默认运行模式。

◆ 检验测试脚本是否存在语法错误，使用 Debug(调试)模式。

◆ 更新已创建的用户界面检查点(GUI checkpoint)和位图检查点(bitmap checkpoint)的预期值，使用 Update(校正)模式。

5. 查看测试结果

一次测试运行结束以后，在 WinRunner 的测试结果窗口，WinRunner 将不同运行结果标以不同颜色(绿色的显示表示运行正确结束，红色的显示表示运行失败)，测试人员能很快得知该测试是成功还是失败，也可以通过“Tools”菜单中的子菜单“Test Results...”获得测试结果，如图 16.10 所示。如果在测试结果中有红色显示，可以双击该红色结果，就会弹出错误产生的原因供测试人员参考。

注意：WinRunner 可以设定针对应用程序响应的等待时间。默认的等待时间是 10 秒，如果在这期间应用程序来不及响应，测试运行也可能失败。这些可以通过延长等待时间或设置同步点予以解决。

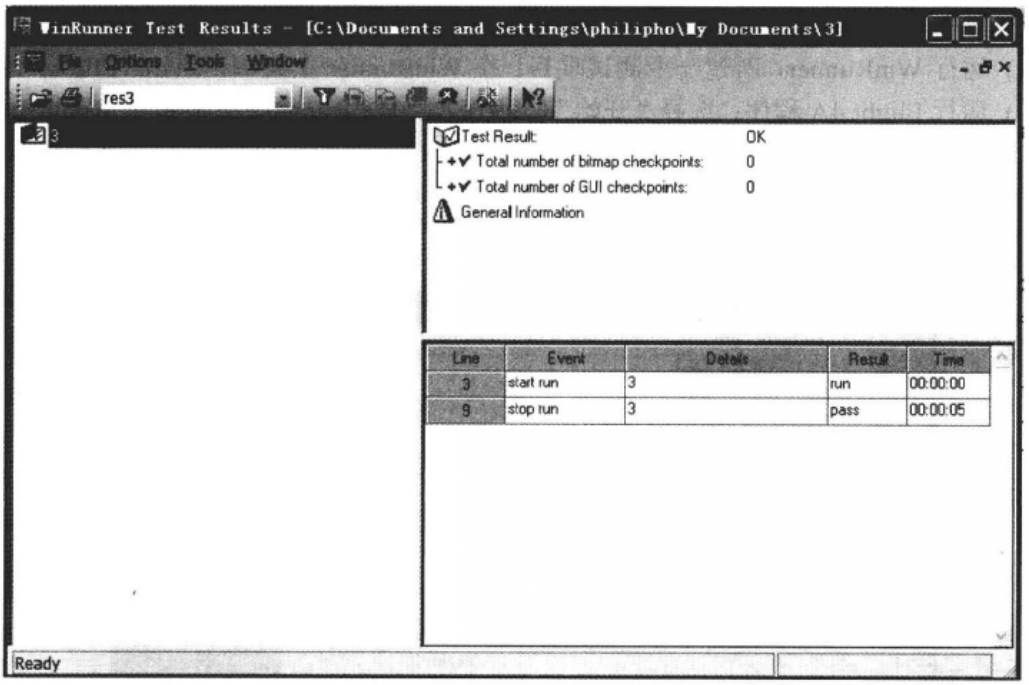


图 16.10　WinRunner 测试结果界面

16.1.5　WinRunner 测试示例二

16.1.4 节的例子使读者初步认识了 WinRunner 的测试过程和方法，但是，怎么样判断程序的对错呢？答案是：在执行了某个步骤时，判断程序的返回结果与预期结果是否一致。

WinRunner 可以在执行了某个步骤后，检查某个对象的单个或者多个属性是否与预先定义的相一致，或者是多个对象的多个属性是否与预先定义的相一致，又或者是将对象作为图片，比较是否与预期的相一致。对于数据库应用，则检查某个表或是某个查询的记录内容、数量等是否与预期的相一致。

WinRunner 使用 GUI 检查点来检验被测软件中的 GUI 对象。比如可以检查一个对话框在何时打开，一个 button 是否可用等。用户所要做的就是指向这个对象，选择想要 WinRunner 检查的属性。可以检查 WinRunner 建议的属性或自己指定属性。GUI 对象和被选定的属性保存在一个检查清单上，然后 WinRunner 捕捉对象的当前属性值并保存起来作为期望值，这时一个 GUI 检查点就自动被插入到脚本中。在脚本中这个检查点显示为 obj_check_gui 或 win_check_gui 语句。

执行测试时，检查点就把实际值和期望值进行比较。如果不符合就说明检查失败。检查结果可以在测试结果窗口看到。

将 WinRunner 自带的 Flight 4A 设为预期目标，Flight 4B 设为有错误的实际程序。具体步骤如下。

1. 创建 GUI map

使用 GUI Map file per test 模式，可省去这一步骤。

2. 创建测试

(1) 运行 WinRunner，新建一个测试项目：在 WinRunner 主菜单栏点击“File”|“New”。

(2) 运行 Flight 4A 程序：选择“开始”|“程序”|“WinRunner”|“Sample Applications”，点击“Flight 4A”快捷方式，显示出示例程序的登录界面。在登录界面，点击“Agent Name”输入框，输入“admin”，点击“Password”输入框，输入“mercury”，点击“OK”按钮，程序登录成功。示例程序主界面如图 16.11 所示。

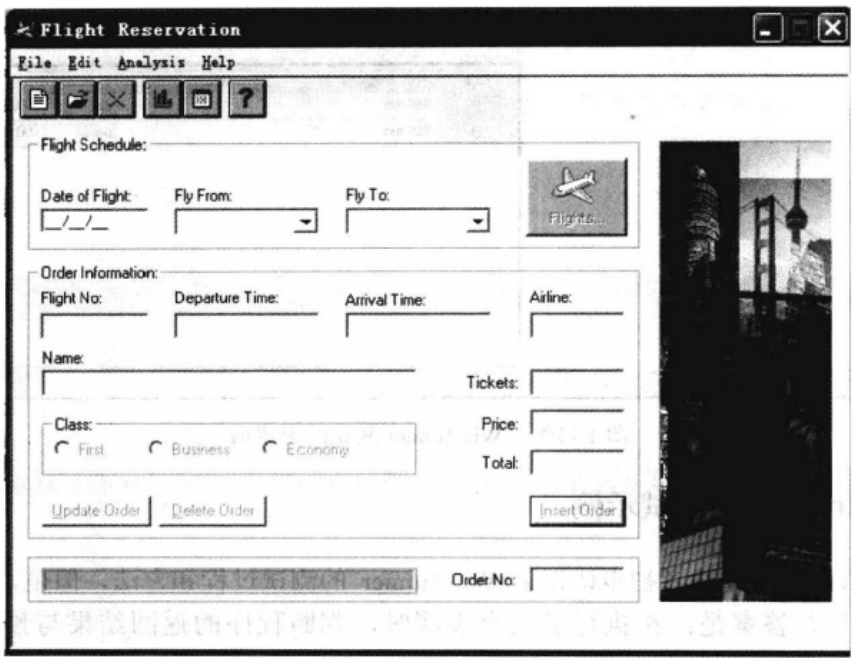


图 16.11 示例程序 Flight 4A 主界面

(3) 开始录制：在 WinRunner 主菜单栏点击“Record”按钮，WinRunner 进入录制状态。

(4) 进入 Flight 4A “Open Order”窗口：在 Flight 4A 示例程序主界面，点击“File”|“Open Order”，如图 16.12 所示。

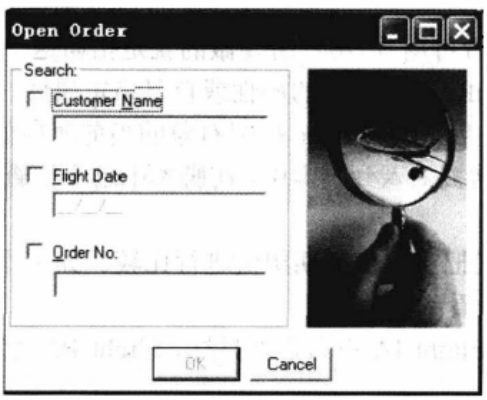


图 16.12 示例程序 Flight 4A 的“Open Order”主界面

(5) 点击”“Order No.”按钮，“Customer Name”复选框应置灰，如图 16.13 所示。

图 16.13　Order No.界面

(6) 检查“Customer Name”复选框的 enable 属性：点击 WinRunner 菜单的“Insert”|“GUI Checkpoint”|“For Single Property”按钮，进入选取目标状态，如图 16.14 所示。

图 16.14　WinRunner 选取检查属性设置界面

(7) 移动光标，在 Flight 4A 示例程序“Customer Name”复选框上单击左键，显示检查属性窗口，如图 16.15 所示。

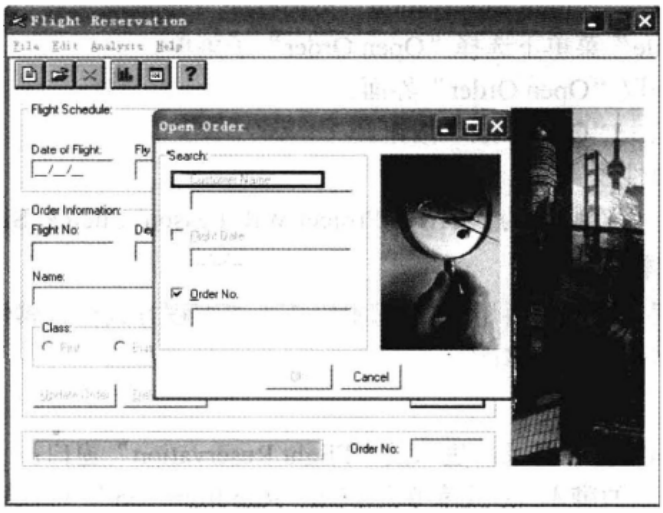


图 16.15　WinRunner 检查属性选取界面

(8) 此时可以看到要检查的属性为“enabled”，确认预期值为 0(相当于 False)，如图 16.16 所示。

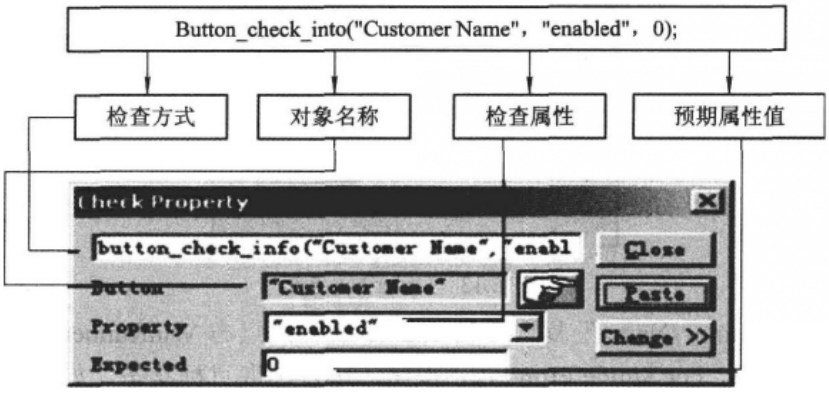


图 16.16 WinRunner 检查属性设置界面

(9) 在 WinRunner 检查属性窗口点击“Pasete”按钮，可以看到脚本中增加了一行语句：

```
button_check_info("Customer Name", "enabled", 0);
```

(10) 停止录制：在 Flight 4A 程序点击“Cancel”按钮返回至主界面；在 WinRunner 主菜单栏点击“Stop”按钮，可以看到 WinRunner 中记录的脚本如下：

```
# Flight Reservation                                    1
set_window ("Flight Reservation", 2);                   2
menu_select_item ("File;Open Order...");                3
                                                        4
Open Order                                              5
set_window ("Open Order", 2);                           6
button_set ("Order No.", ON);                           7
button_check_info("Customer Name", "enabled", 1);       8
```

第一行：注释。

第二行：获取登录“Flight Reservation”示例程序主界面。

第三行：在“File”菜单下选择“Open Order”子菜单。

第六行：用户获取“Open Order”界面。

第七行：用户点击“Order No.”按钮。

第八行：检查属性设置。

(11) 保存脚本：保存脚本为“C:\TestProject\WR_Lesson_Cherck_Single”。

3. 修改测试脚本

用户还可以根据自己的需要用 TSL 脚本语言手工编程方式检查属性，修改测试脚本达到测试的要求，在本例中不做修改。

4. 执行测试

(1) 确认 Flight 4A 程序登录后进入的“Flight Reservation”窗口；

(2) 回放刚才录制的脚本，在主菜单栏点击“Run from Top”，可以看到刚刚所做的操作均被重现了一遍。

5．查看测试结果

当一次测试运行结束以后，在 WinRunner 的测试结果窗口可以看到没有错误，全部通过，如图 16.17 所示。

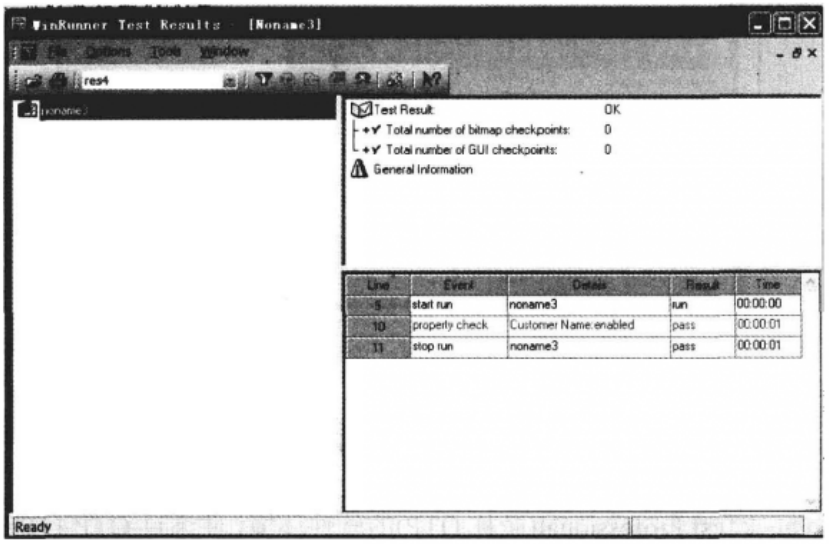


图 16.17　WinRunner 测试结果界面

6．执行测试

(1) 确认退出 Flight 4A 程序。

(2) 确认 Flight 4B 程序登录后进入的“Flight Reservation”窗口(参考 Flight 4A 程序的启动)。

(3) 回放刚才录制的脚本，在 WinRunner 主菜单栏中点击“Run from Top”，可以看到执行过程中出现了错误。点击“Order No.”按钮，“Customer Name”复选框没有置灰，如图 16.18 所示。

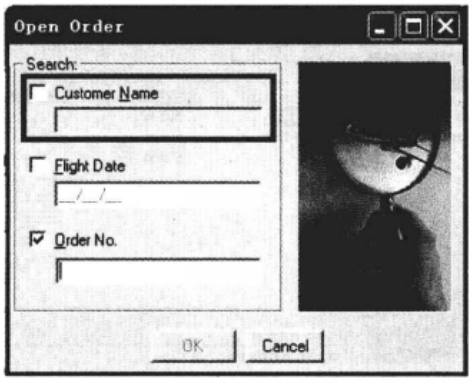


图 16.18　示例程序 Flight 4A 点击 Order No.界面

7．查看测试结果

在 WinRunner 的测试结果窗口可以看到检查点的记录为红色，结果为 fail，双击后可以看到原因为与预期的结果不一致，如图 16.19 所示。

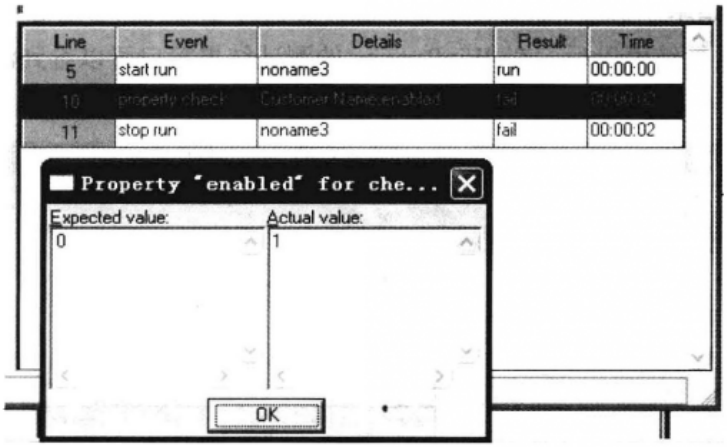


图 16.19 WinRunner 测试结果界面

16.2 QuickTest Professional 简介

本节介绍 QuickTest Professional(简称 QTP)的软件环境，并通过 QTP 自带的飞机订票网站实例讲解了 QTP 的使用方法。

16.2.1 认识 QuickTest Professional 工作环境

1. 启动 QuickTest Professional

点击“开始”|“程序”|“QuickTest Professional”|“QuickTest Professional”，启动 QTP。

在 QuickTest Professional 启动时，可以选择支持 ActiveX、VisualBasic 或 Web 的插件。其它插件需要单独再向 MI 公司购买，建议不要同时载入所有的插件，不必要的插件可能会对录制或执行造成问题。本节例子选用 Web 插件，如图 16.20 所示。

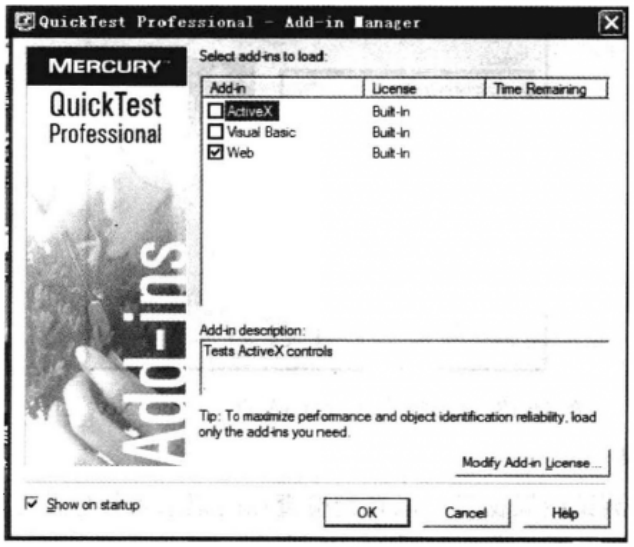


图 16.20 QTP 启动插件选择界面

2. QuickTest Professional 主界面介绍

图 16.21 显示了录制测试后将出现的 QTP 窗口，其中显示了除“调试查看器”窗格以外的所有工具栏和窗格。

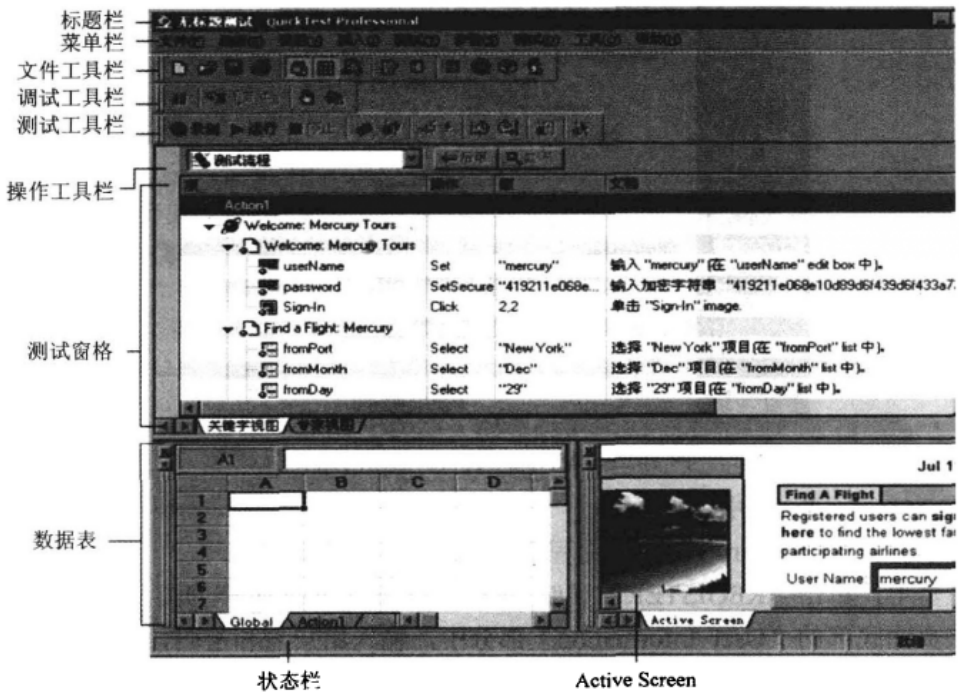


图 16.21　QTP 测试主界面

16.2.2　QTP 测试示例

用户安装 QTP 后，会自动安装单机版的 Flight Reservation(航班预订)软件，作为用户测试应用程序的范例软件。

下面通过 QTP 自带的飞机订票网站示例，讲解 QTP 的使用过程。具体步骤如下所示：

1. 准备录制

● 优化测试的浏览器设置

使用 Internet Explorer 作为浏览器，应清除用户名和密码的“自动完成”选项。这样将确保在创建测试时，可以精确录制所有的操作。

要清除“自动完成”选项，请执行下列操作：

(1) 在 Internet Explorer 的菜单栏中，选择“工具”→“Internet 选项”→“内容”选项卡；

(2) 在“个人信息”区域中单击“自动完成”，将打开“自动完成设置”对话框；

(3) 在“自动完成功能应用于”区域中，清除“表单上的用户名和密码”选项；

(4) 单击“确定”保存更改并关闭“自动完成设置”对话框，然后再次单击“确定”，关闭“Internet 选项”对话框。

● 启动 Mercury Tours 应用程序

在 Web 浏览器中，键入以下 URL：http://newtours.demoaut.com，将打开 Mercury Tours

主页，如图 16.22 所示。

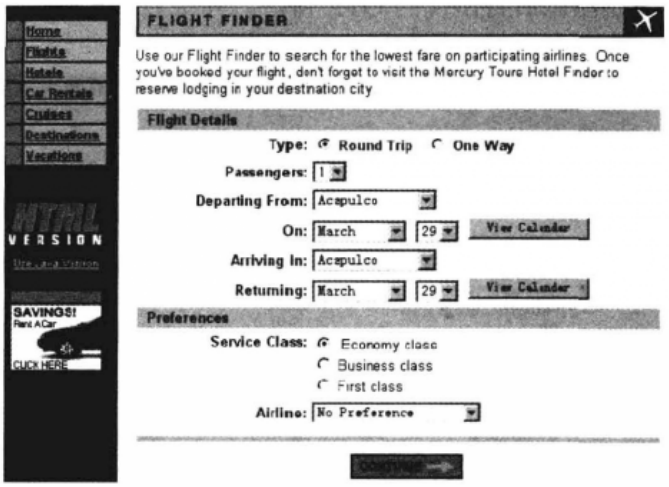


图 16.22 Mercury Tours 范例网站界面

● 在 Mercury Tours 中注册

要登录并使用 Mercury Tours 应用程序，必须成为注册用户。

在主页上，单击“REGISTER”导航按钮，将打开“Register”页。

在该页面底部的“User Information”部分中，输入用户名和密码，并确认密码。(其他信息是可选的。)

单击“SUBMIT”，Mercury Tours 将确认注册。在确认页上，单击“SIGN-ON”导航按钮进入应用程序，将打开“Sign-on”页。输入已注册的用户名和密码，然后单击“SUBMIT”。

此时将打开“Flight Finder”页。

● 浏览 Mercury Tours 站点

从“Flight Finder”页开始，按照屏幕上的说明获得航班信息并预定航班。

● 结束 Mercury Tours 会话

在浏览 Mercury Tours 应用程序完成后，单击“Flight Confirmation”页上的“LOG OUT”按钮，或单击任何应用程序页顶部的“SIGN-OFF”链接。

再次登录，只需在“Sign-on”页或 Mercury Tours 主页中输入已注册的用户名和密码即可。

● 关闭 Web 浏览器

此时，便可以开始使用 QTP 在 Mercury Tours 应用程序上创建测试。

2. 录制测试脚本

通过 QTP 录制一个测试脚本，内容为在 Mercury Tours 范例网站上预定一张从纽约(New York)到旧金山(San Francisco)的机票。

(1) 启动 QTP：请选择“开始”|“程序”|“QuickTest Professional”|“QuickTest Professional”。在“加载项管理器”中，确认 Web 加载项处于选定状态，并清除所有其它加载项。单击“确定”，关闭“加载项管理器”并打开 QTP。

(2) 新建一个测试项目：在主菜单栏点击“File”|“New”。

(3) 开始录制测试脚本：点选工具栏上的“Record”按钮，打开“Record and Run Settings”对话窗口，如图 16.23 所示；在“Web”标签页选择“Open the following address when a record or run session begins”，在“Type”下拉列表中选择“Microsoft Internet Explorer”为浏览器的类型；在“Address”中添加“http:// newtours.demoaut.com/”(网站地址)，这样，在录制的时候，QTP 会自动打开 IE 浏览器并连接到 Mercury Tours 范例网站上。

(4) 切换到“Windows Application”标签页，如图 16.24 所示。如果选择“Record and run test on any open Windows-based application”单选按钮，则在录制过程中 QTP 会记录对所有的 Windows 程序所做的操作。如果选择“Record and run on these application(opened when a session begins)”单选按钮，则在录制过程中，QTP 只会记录对那些添加到下面“Application details”列表框中的应用程序的操作(可以通过“Add”、”“Edit”、”“Delete”按钮来编辑这个列表)。

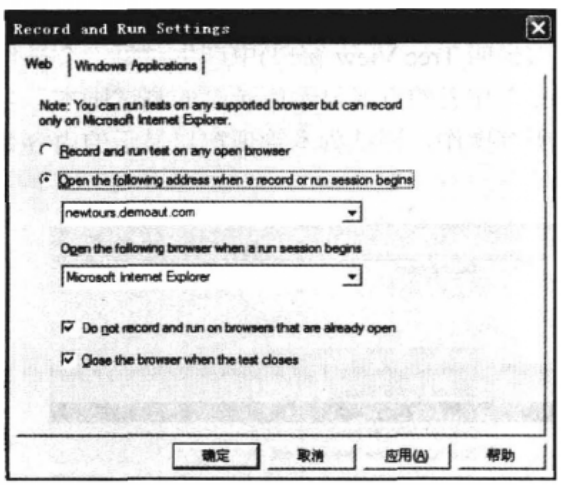


图 16.23　QTP 录制脚本设置界面

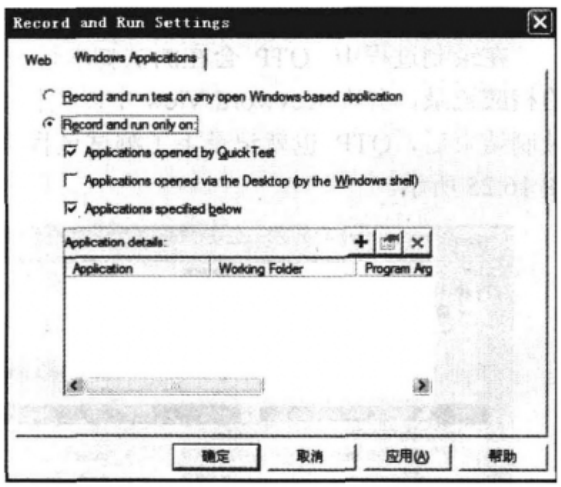


图 16.24　QTP 录制脚本设置界面

(5) 选择第二个单选按钮。因为只是对 Mercury Tours 范例网站进行操作，不涉及到 Windows 程序，所以保持列表为空。

(6) 点击“确定”按钮，开始录制，将自动打开 IE 浏览器并连接到 Mercury Tours 范例网站上。

(7) 登录 Mercury Tours 网站，在用户名和密码栏中输入注册时使用的账号和密码，点击“Sign-in”，进入“Flight Finder”网页。

(8) 输入以下订票数据：

Departing From：New York

On：May 14

Arriving In：San Francisco

Returning：May 28

Service Class：Business class

其它字段保留默认。

点击“CONTINUE”按钮打开“Select Flight”页面。

(9) 选择飞机航班：可以保存默认值，点击“CONTINUE”按钮打开“Book a Flight”

页面。输入必填字段(红色字段)，输入用户名和信用卡号码(信用卡可以输入虚构的号码，如 8888-8888)。点击网页下方的“SECURE PURCHASE”按钮，打开“Flight Confirmation”网页，完成定制流程。

(10) 查看订票数据，并选择“BACK TO HOME”回到 Mercury Tours 网站首页。

(11) 停止录制：在 QTP 工具列上点击“Stop”按钮，停止录制。到这里已经完成了预定从“纽约—旧金山”机票的动作，并且 QTP 已经录制了从按下“Record”按钮到“Stop”按钮之间的所有操作。

(12) 保存脚本：点击工具栏上的“Save”按钮，开启“Save”对话窗口。选好路径，填写文件名，取名为 Flight，点击“保存”按钮进行保存。

通过以上几个步骤，已经录制了一个完整的测试脚本——预定从纽约到旧金山的机票。

3. 分析测试脚本

在录制过程中，QTP 会在测试脚本管理窗口(也叫 Tree View 窗口)中产生对每一个操作的相应记录，并在 Keyword View 中以类似 Excel 工作表的方式显示所录制的测试脚本。当录制结束后，QTP 也就记录下了测试过程中的所有操作。测试脚本管理窗口显示的内容如图 16.25 所示。

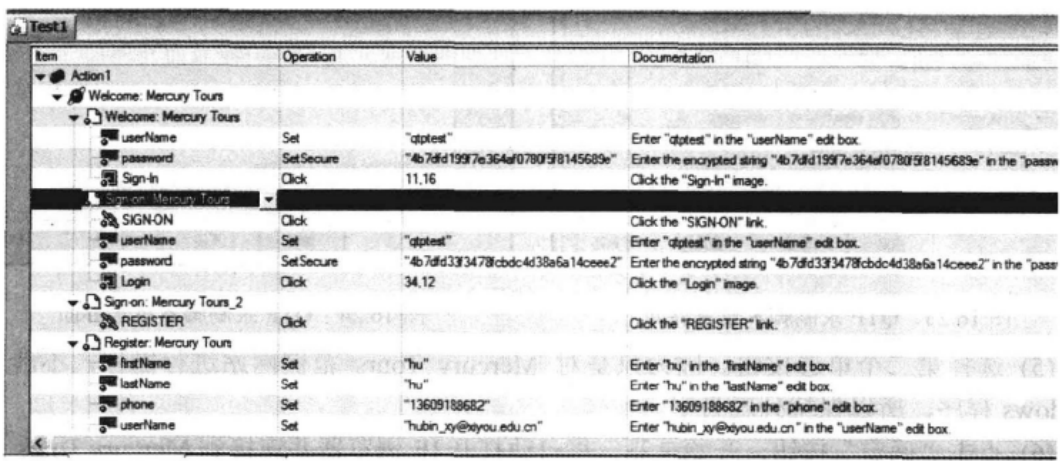


图 16.25 QTP 脚本管理窗口

在 Keyword View 中的每一个字段都有其意义：

◆ Item：以阶层式的图标表示这个操作步骤所作用的组件(测试对象、工具对象、函数呼叫或脚本)。

◆ Operation：要在这个作用到的组件上执行的动作，如点击、选择等。

◆ Value：执行动作的参数，例如当鼠标点击一张图片时是用左键还是右键。

◆ Assignment：使用到的变量。

◆ Comment：在测试脚本中加入的批注。

◆ Documentation：自动产生用来描述此操作步骤的英文说明。

脚本中的每一个步骤在 Keyword View 中都会以一列来显示，其中用来表示此组件类别的图标以及步骤的详细数据。

图 16.26 是针对一些常见的操作步骤的详细说明。

步骤			描述
Action1			Action1 是操作名。
Welcome: Mercury Tours			浏览器调用 Welcome:Mercury Tours 网站。
Welcome: Mercury Tours			Welcome:Mercury Tours 是网页的名称。
userName	Set	"mercury"	userName 是编辑框的名称。Set 是在编辑框上执行的方法。mercury 是编辑框的值。
password	SetSecure	"4082820183...	password 是编辑框的名称。SetSecure 是在编辑框上执行的加密方法。4082820183afe512e8bc91c1f7222dbd 是 password 的加密值。
Sign-In	Click	2,2	Sign-In 是图像链接的名称。Click 是在图像上执行的方法。2, 2 是图像单击位置的 x 坐标和 y 坐标。

图 16.26　QTP 操作步骤说明

4．执行测试

当运行录制好的测试脚本时，QTP 会打开被测试程序，执行在测试中录制的每一个操作。测试运行结束后，QTP 显示本次运行的结果。接下来，执行在上一节中录制的 Flight 测试脚本。

(1) 打开录制的 Flight 测试脚本。

(2) 设置运行选项。点击“Tool”|“Options”，打开设置选项对话框。选择“Run”标签页，如图 16.27 所示，如果要将所有画面储存在测试结果中，在 Save still image captures to results”选项中选择“Always”选项。一般情况下选择“For error”或“For errors and warning”表示在回放测试过程中出现问题时，才保存图像信息。在这里为了更多地展示 QTP 的功能，选择使用“Always”选项。

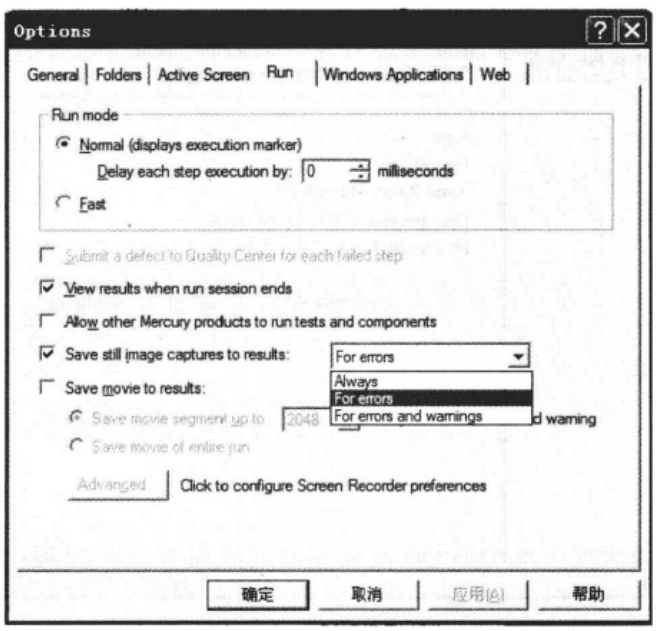


图 16.27　QTP Options 设置

(3) 在工具条上点击“Run”按钮，打开“Run”对话框，询问要将本次的测试运行结果保存到何处。选择“New run results folder”单选按钮，设定好存放路径(在这使用预设的测试结果名称)，点击“确定”按钮开始执行测试，如图 16.28 所示。

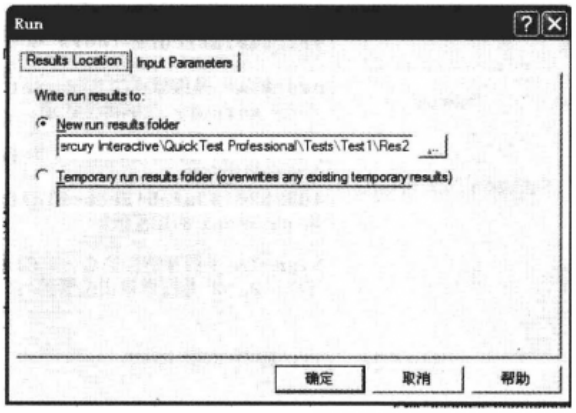


图 16.28 QTP Run 设置

(4) 此时可以看到 QTP 按照在脚本中录制的操作，一步一步地运行测试，操作过程与手工操作时完全一样。同时可以看到在 QTP 的 Keyword View 中会出现一个黄色的箭头，指示目前正在执行的测试步骤。

(5) 如果在执行测试的时候出现错误，会显示一个错误信息对话框。

5. 查看测试结果

在测试执行完成后，QTP 会自动显示测试结果窗口，如图 16.29 所示。

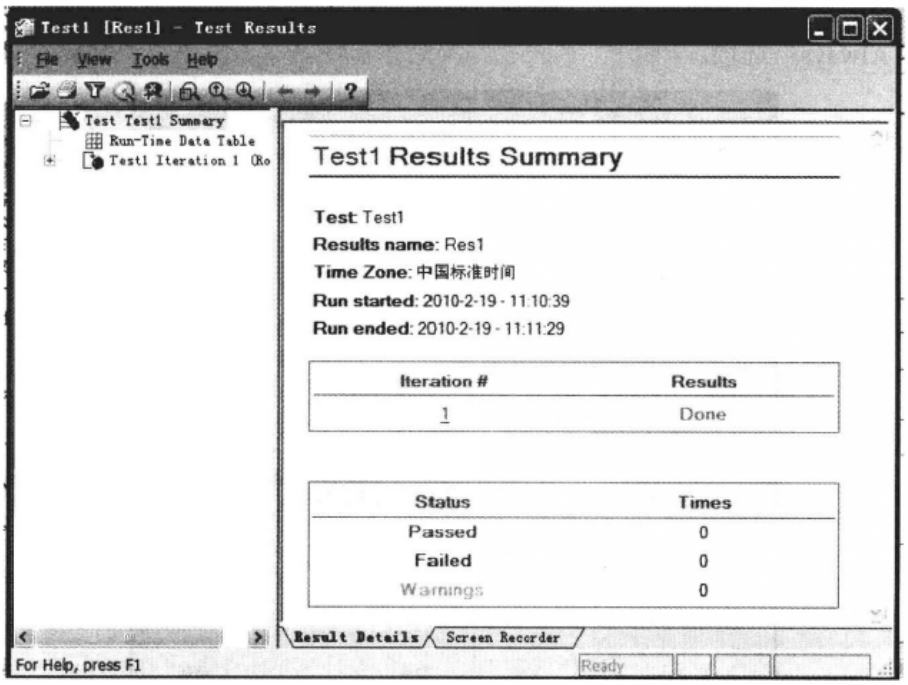


图 16.29 QTP 测试结果界面

在这个测试结果窗口中，分两个部分显示测试执行的结果。

◆ 左边显示 Test results tree，以树视图的方式显示测试脚本所执行的步骤。可以选择“+”检查每一个步骤，所有的执行步骤都会以图示的方式显示。可以设定 QTP 以不同的资料执行每个测试或某个动作，每执行一次反复称为一个迭代，每一次迭代都会被编号(在上面的例子中只执行了一次迭代)。

◆ 右边则是显示测试结果的详细信息。在第一个表格中显示哪些迭代是已经通过的，哪些是失败的。第二个表格显示测试脚本的检查点，哪些是通过的，哪些是失败的，以及有几个警告信息。

在上面的测试中，所有的测试都是通过的，在脚本中也没有添加检查点。接下来查看 QTP 执行测试脚本的详细结果，以及选择某个测试步骤时出现的详细信息。

在树视图中展开“Flight Iteration 1(Row 1)”|“Action1 Summary”|“Welcome MercuryTours”|“Find a Flight: Mercury”，选择“fromPost：Select ”|“New York”。

在这个测试结果窗口中显示三个部分，分别是：

◆ 左边是 Test results tree。展开树视图后，显示了测试执行过程中的每一个操作步骤。选择某一个测试步骤，会在右边区域显示相应的信息。

◆ 右上方是 Test results detail。对应当前选中的测试步骤，显示被选取测试步骤执行时的详细信息。

◆ 右下方是 Active Screen。对应当前选中的测试步骤，显示该操作执行时应用程序的屏幕截图。当选中 test results tree 上的网页图示时，会在“Active Screen”中看到执行时的画面。当选中 test results tree 上的测试步骤(在某个对象上执行某个动作)时，除了显示当前的画面外，对象还会被粉色的方框框住。上面的例子中，在“Active Screen”中点击被框住的“DepartingFrom”下拉菜单，会显示其它的选项，如图 16.30 所示。

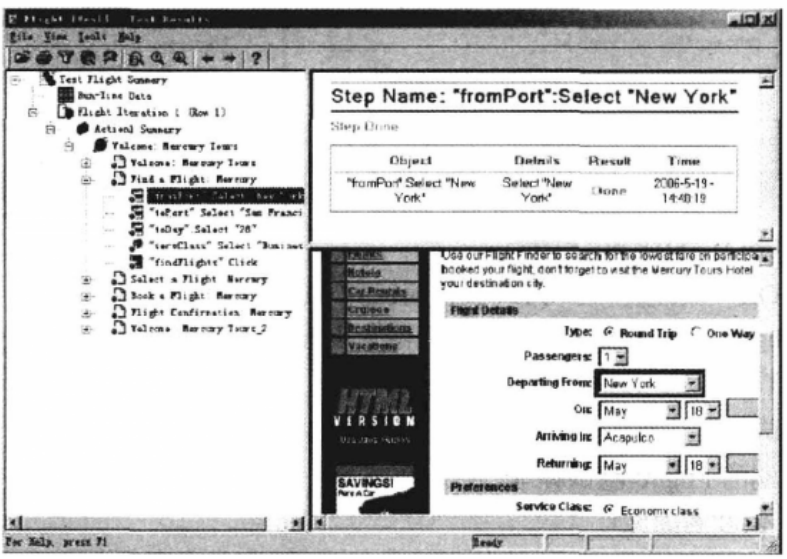


图 16.30　QTP 测试结果界面

6．设置检查点

“检查点”是将指定属性的当前值与该属性的期望值进行比较的验证点。这能够确定网站或应用程序是否正常运行。当添加检查点时，QTP 会将检查点添加到关键字视图中的

当前行并在专家视图中添加一条“检查点”语句。运行测试或组件时，QTP 会将检查点的期望结果与当前结果进行比较。如果结果不匹配，检查点就会失败。可以在“测试结果”窗口中查看检查点的结果。

打开 Flight 测试脚本，将脚本另存为“Checkpoint”测试脚本。在 Checkpoint 测试脚本中创建一个文字检查点，检查在“Flight Confirmation”网页中是否出现“New York”。

(1) 确定要建立检查点的网页：展开“Action1”|“Welcome: Mercury Tours”，选择“Flight Confirmation: Mercury”页面，在“Active Screen”中会显示相应的页面。

(2) 建立文字检查点：在“Active Screen”中选择“Departing”下方的“New York”。对选取的文字单击鼠标右键，并选取“Insert Text Checkpoint”，打开“Text Checkpoint Properties”对话窗口。当“Checked Text”出现在下拉式清单中时，“Constant”字段显示的就是选取的文字，这也就是 QTP 在执行测试脚本时所要检查的文字，如图 16.31、图 16.32 所示。

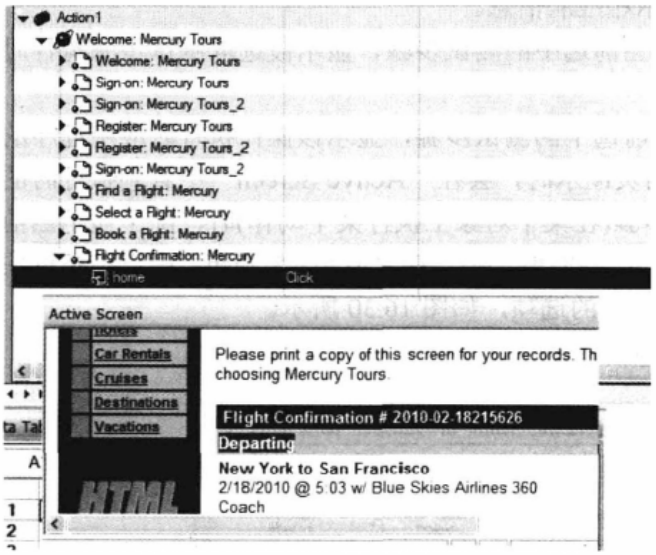


图 16.31 QTP 加入检查点

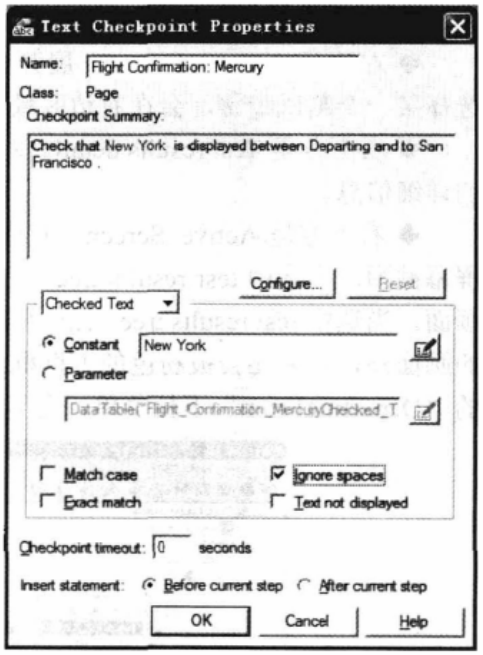


图 16.32 QTP 检查点设置

(3) 点击“OK”关闭窗口。QTP 会在测试脚本中添加一个文字检查点，这个文字检查点会出现在“Flight Confirmation: Mercury”网页下方。

(4) 在工具栏上点击“Save”，保存脚本。

7．执行测试

(1) 打开录制的 Flight 测试脚本。

(2) 运行脚本(参考上文)。

8．查看测试结果

在 Test results tree 中展开“Checkpoint Iteration 1(Row 1)”|“Action1 Summary”|“Welcome:Mercury Tours”|“Flight Confirmation: Mercury”，并选择“Checkpoint " New York " ”，显示如图 16.33。因为文字检查点的实际值与预期值相同，所以检查点的结果为 Passed。

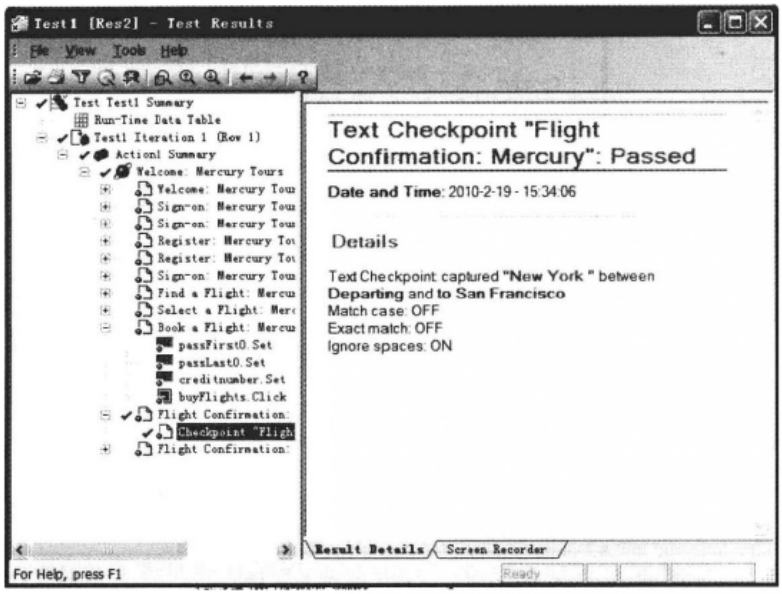


图 16.33　QTP 检查点测试结果界面

9. 参数化

在测试脚本中，出发地点“纽约”是个常数值，也就是说，每次执行测试脚本预定机票时，出发地点都是纽约。现在，将测试脚本中的出发地点参数化，这样，执行测试脚本时就会以不同的出发地点去预定机票了。

(1) 首先，打开 Checkpoint 测试脚本，将脚本另存为“Parameter”，然后选择要参数化的文字：在视图树中展开“Action1”|“Welcome: Mercury Tours”|“Find a Flight: Mercury”，如图 16.34 所示。

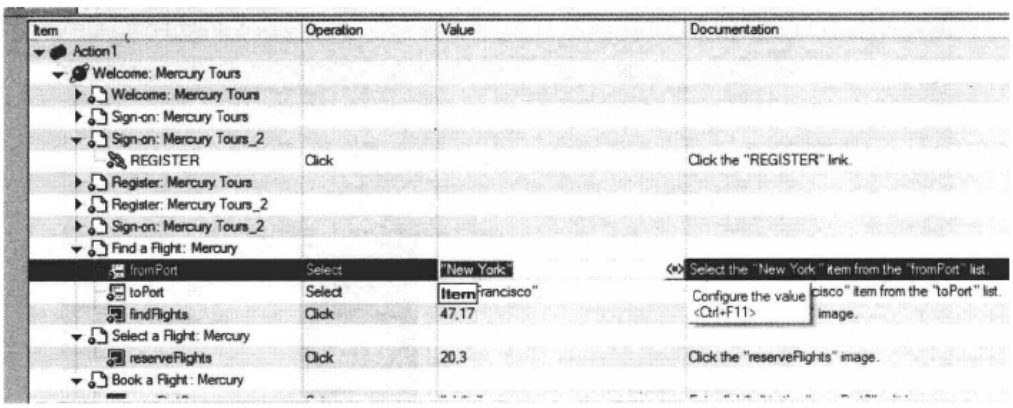


图 16.34　QTP 脚本管理加入参数化

(2) 在视图树中选择“fromPort”右边的“Value”字段，然后再点击参数化图标，开启“Value Configuration Options”对话窗口。

(3) 设置要参数化的属性，选择“Parameter”选择项，这样就可以用参数值来取代“NewYork”这个常数了。在参数中选择“Data Table”选项，这样这个参数就可以从 QTP 的 Data Table 中取得，将参数的名字改为“p_Item”，如图 16.35 所示。

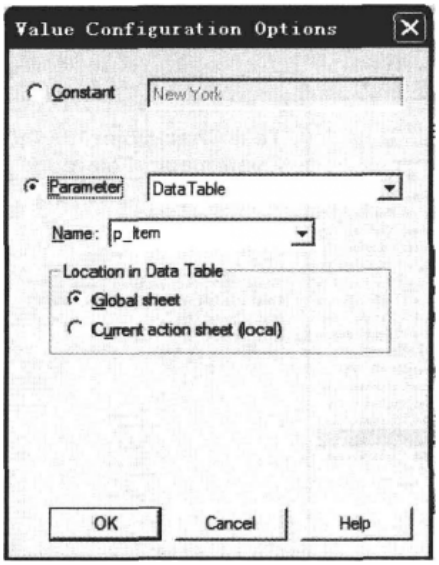


图 16.35 QTP 操作工具栏

(4) 点击“OK”确认，QTP 会在 Data Table 中新增 departure 参数字段，并且插入了一行 New York 的值，“New York”会成为测试脚本执行使用的第一个值。

(5) 参数化以后可以看到树视图中的变化，如图 16.36 所示。在参数化之前，这个测试步骤显示“foomPost …Select… New York”，现在，这个步骤变成了“foomPost …Select… Data Table(departure, dtGlobalSheet)”。而且当点击 Value 字段时，Value 字段会显示如图 16.36 所示，表示此测试步骤已经被参数化，而且其值从 Data Table 的 departure 字段中获得。在 departure 字段中加入出发地资料，使 QTP 可以使用这些资料执行脚本。在 departure 字段的第二行、第三行分别输入 Portland、Seattle。

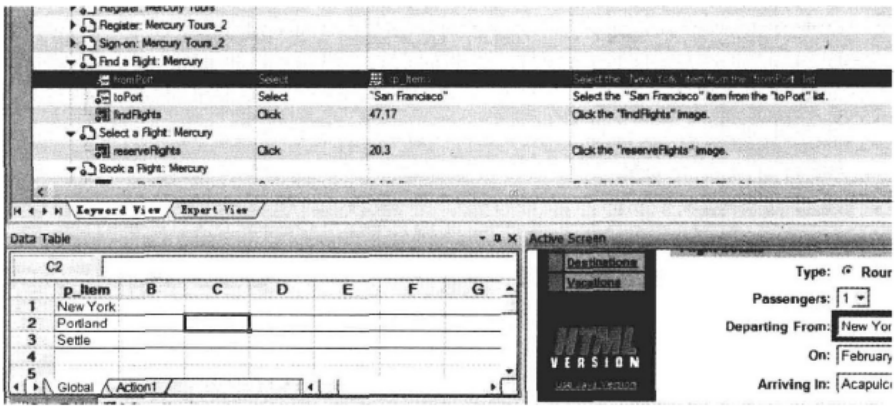


图 16.36 QTP 脚本界面-参数化后

(6) 保存测试脚本。

(7) 修正受到参数化影响的步骤：当测试步骤被参数化以后，可能会影响到其它的测试步骤也要参数化。例如为了验证在“Flight Confirmation”网页中是否出现“New York”(第三章创建文字检查点)，在网页上添加了一个文字检查点。那么，就要对出发地的文字检查点作参数化，以符合对出发地参数化的预期结果。

(8) 修正文字检查点，首先在树视图中展开“Action1”|“Welcome: Mercury Tours”|“Flight Confirmation: Mercury”页面，然后点击鼠标右键，选择“Checkpoint Properties”，打开“TextCheckpoint Properties”对话窗口，如图 16.37 所示。

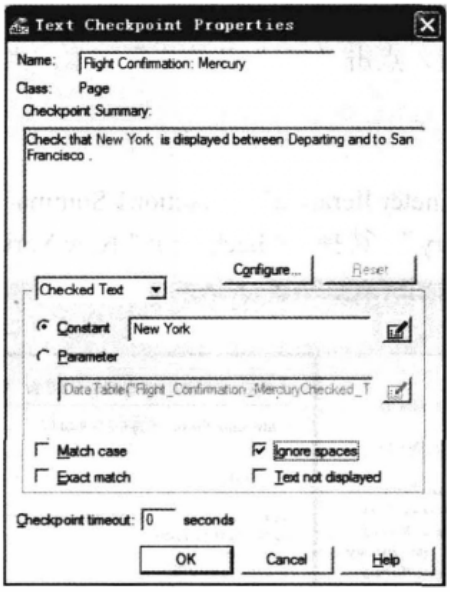


图 16.37　QTP 修正文字检查点

(9) 在“Checked Text”的 Constant 字段中显示为“New York”，表示测试脚本在每次执行时，这个文字检查点的预期值都为“New York”。选择 Parameter，点击旁边的“Parameter Options”按钮，打开“Parameter Options”对话窗口：在参数类型选择框选择“Data Table”选项，在名字选择框选择“p_Item”选项，指明这个文字检查点使用 p_Item 字段中的值作为检查点的预期值，如图 16.38 所示。

图 16.38　QTP 修正文字检查点

(10) 点击“OK”关闭窗口，这样文字检查点也被参数化了。

10. 执行测试

参数化测试脚本后，运行 Parameter 测试脚本。QTP 会使用 Data Table 中的 p_Item 字段值，执行三次测试脚本。

执行测试脚本：点击工具栏上的“Run”按钮，开启 Run 对话窗口，选取“New run resultsfolder”，其余为默认值，点击“OK”开始执行脚本。当脚本运行结束后，会开启测试结果窗口。

11. 查看测试结果

在树视图中，展开“Parameter Iteration2”|“Action1 Summary”|“Welcome MercuryTours”|“Flight Confirmation: Mercury”，选择“Checkpoint " New York " ”，显示如图 16.39 所示。

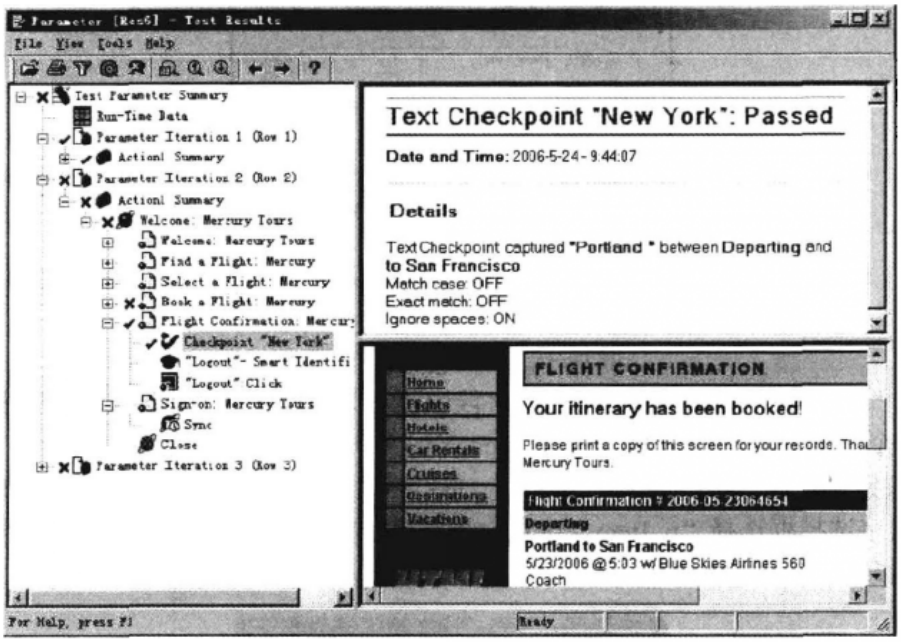


图 16.39 QTP 测试结果参数化后

在检查点“Details”窗口中，显示预期要被记录的 Portland 的值，所以文字检查点为通过。同时也可以看到在下方的“Application”窗口中，显示机票的出发地点也是 Portland。

在图 16.39 中可以看出，虽然每次执行时，文字检查点的结果都是通过的，但是第二次与第三次的执行结果仍然为失败。这是因为出发地点的改变，造成在表格检查点中的机票价钱改变，导致表格检查点失败。

第 17 章　计算机认证考试

本章介绍了计算机认证考试。从考试的大纲、题型等多个方面对全国四级软件测试工程师考试进行了详细的说明。

17.1　计算机认证考试概述

进入 20 世纪 90 年代以来，西方各国着力普及计算机知识，各国全方位、多层次地开展全国范围的定期的计算机各类等级考试。如美国最权威的教育考试中心 ETS(Educational Testing Service)就面向美国社会推出了“计算机文化考试”、“高级就业计算机科学考试”和“计算机专业领域考试”等三类考试。又如美国计算机专业人员认证学会 ICCP(Institute Certification of Computer Professionals)也实施了有关的认证考试。在英国，由英国计算机学会 BCS(British Computer Society)和 IDPM(Institute of Data Processing Management)分别组织计算机考试，并将其普及到英联邦及其它国家。在日本，自 1969 年开始设立“信息处理技术人员考试”，已经成为仅次于日本大学全国统一考试的第二大规模的全国性考试。

近年来，我国大力设立或引进了各类计算机考试。这些考试主要分两类，一类为国家考试，例如，由原国家人事部和原电子工业部组织的“中国计算机软件专业技术资格和水平考试”、由原国家劳动部组织的“全国计算机及信息高新技术考试”、由原国家教委组织的“全国计算机应用技术等级考试”、由原劳动部职业技能鉴定中心组织的“国家级 INTERNET 证书培训考试”、由原国家教委从剑桥大学引入的“剑桥信息技术(CIT)证书考试”及“全国信息应用技术证书(NIT)考试”等。此外，还有“NOVELL(网络师)认证考试”以及各地高校组织的高校计算机等级考试等。

另一类为厂方认证，如国际 IT 厂方认证的 SUN JAVA 认证证书——JAVA，微软认证证书——MCSE MCP、MCDBA、MCSE，思科认证证书——CCNP、CCNA，AutoCAD 工程师认证证书，CIW 网络安全认证证书，Adobe 中国认证设计师——ACCD，等等。

17.2　各类计算机认证考试

根据参加考试的人数、考试合格证书的效力以及社会对考试的认同程度，计算机认证考试中最有影响力的有以下四种：

(1) 中国计算机软件专业技术资格和水平考试(以下简称水平考试)；

(2) 全国计算机应用技术等级考试(以下简称等级考试)；

(3) 全国信息应用技术考试(以下简称 NIT)；

(4) 全国计算机及信息高新技术考试(以下简称技术考试)。

下面是有关这四种考试的比较。

1. 考试制度

水平考试是由国家人事部组织的，只在每年十月的第二个星期天举行一次考试，由全国统一规定、统一考纲、统一试题、统一时间、统一评分标准及统一合格录取标准，而且只有笔试，考试合格颁发合格证书；等级考试和水平考试同样是全国“统考”，所不同的是它是由国家教育部组织的，考试在每年上、下半年各考一次，考试方式分笔试及上机考试两项；NIT 和技术考试则侧重于对应试者能力的检测，前者是由国家教育部组织的，后者则是由国家劳动和社会保障部组织的，两种考试都是随训随考，没有固定的时间，考试方式灵活多样。

2. 证书的类别与效力

在这四种计算机认证考试中，只有等级考试将证书分为两类，一类是普通的合格证书，另一类是优秀证书(所谓“优秀”，是指笔试成绩在 90 分以上、机试成绩则必须满分)。这四种认证考试的证书都是全国有效的。

水平考试的合格证书是职称评定的有力依据，在我国，通常拥有水平考试的合格证书就能找到一份不错的工作，因此水平考试的合格证书取之不易；等级考试的合格证书仅说明持证者通过何种考试，仅供用人单位参考，对电脑操作要求不是特别高的岗位，等级考试合格证书能让单位相信持证者的能力，而且很多部门都要求公务员必须持有等级考试(四级)的合格证书。另外，等级考试证书还有一项其它计算机认证考试所不具备的作用：凡持有等级考试证书者，参加自学考试时可申请免考高数、会计等 22 个专业的“计算机应用基础(A)上机”课程(标准号 3016)，且从 1999 年起，各主考学校不再安排这些专业该课程的考试，该课程的考试成绩均由全国计算机等级考试取得。NIT 技术考试的合格证书只证明持证者达到什么样的操作水平，因此只有寻找与持证者能力相关的工作时才能充分发挥证书的作用。

这四种计算机认证考试取得证书的难易程度不同。水平考试是在考试成绩出来之后划定合格标准，类似于每年一度的高考，因此对合格人数把关甚严，能拿到合格证书相当不容易；等级考试则是规定了合格分数，没有特殊情况是不会改变合格分数的，只要应试者成绩在合格分数以上，就能拿到合格证书；NIT 是培训过程中考试，直到参加培训的学员将考题做对为止，只要能顺利完成考题即可获得合格证书(需考试中心审核试题完成情况)；技术考试的试题考前就已经发给应试者(在试题集中)，应试者参加考试并达到合格线即可获得证书。除 NIT 是开卷考试外，其它三种均是闭卷考试。

17.3 全国计算机等级考试

1. 考试性质

全国计算机等级考试是经国家教育部批准，由教育部考试中心主办，用于考查应试人员计算机应用知识与能力的等级考试。

2. 组织机构

教育部考试中心聘请全国著名计算机专家组成的“全国计算机等级考试委员会”，负责

设计考试方案，审定考试大纲，制定命题原则，指导和监督考试的实施。教育部考试中心负责实施考试，制定有关规章制度，编写考试大纲及相应的辅导材料，命制试卷、答案及评分标准，研制考试必需的计算机软件，开展考试研究和宣传等。教育部考试中心在各省(自治区、直辖市)设立省级承办机构负责本地考试的宣传、推广和实施，根据规定设置考点，组织评卷，颁发合格证书等。省级承办机构下设考点负责考生的报名、笔试、上机考试及相关的管理工作，并负责发放成绩通知单和转发合格证书。

3. 证书作用

全国计算机等级考试一级证书表明持证人具有计算机的基础知识和初步应用能力，掌握表处理(Word)、电子表格(Excel)和演示文稿(PowerPoint)等办公自动化(Office)软件的使用及因特网(Internet)应用的基本技能，具备从事机关、企事业单位文秘和办公信息计算机化工作的能力。

二级证书表明持证人具有计算机基础知识和基本应用能力，能够使用计算机高级语言编写程序和调试程序，可以从事计算机程序的编制工作、初级计算机教学培训工作以及计算机企业的业务和营销工作。

三级“PC 技术”证书，表明持证人具有计算机应用的基础知识，掌握 Pentium 微处理器及 PC 计算机的工作原理，熟悉 PC 机常用外部设备的功能与结构，了解 Windows 操作系统的基本原理，能使用汇编语言进行程序设计，具备从事机关、企事业单位 PC 机使用、管理、维护和应用开发的能力；三级“信息管理技术”证书，表明持证人具有计算机应用的基础知识，掌握软件工程、数据库的基本原理和方法，熟悉计算机信息系统项目的开发方法和技术，具备从事管理信息系统项目和办公自动化系统项目开发和维护的基本能力；三级“数据库技术”证书，表明持证人具有计算机应用的基础知识，掌握数据结构、操作系统的基本原理和技术，熟悉数据库技术和数据库应用系统项目开发的方法，具备从事数据库应用系统项目开发和维护的基本能力；三级“网络技术”证书，表明持证人具有计算机网络通信的基础知识，熟悉局域网、广域网的原理及其安全维护方法，掌握因特网(Internet)应用的基本技能，具备从事机关、企事业单位组网、管理及其开展信息网络化的能力。

四级证书表明持证人掌握计算机的基础理论知识和专业知识，熟悉软件工程、数据库和计算机网络的基本原理和技术，具备从事计算机信息系统和应用系统开发和维护的能力。

4. 相关学习网站

(1) 全国计算机等级考试：http://www.ncre.cn/ncre_new/

(2) 全国计算机等级考试官方论坛：http://bbs.ncre.cn/

(3) 全国计算机等级考试网——无忧考试吧：http://www.wyks8.com/ncre/

(4) 全国计算机等级考试(NCRE)网：http://www.51test.net/ncre/

(5) 江苏省计算机等级考试历年真题：http://sjweb.hhit.edu.cn/vbweb/test/

(6) 考试吧——计算机等级考试第一门户 http://www.exam8.com/computer/djks/

(7) 考试试题网——搜集自考、成考、英语计算机及职业考试试题、笔记与信息的网站：http://www.ksstw.com/Main/index.asp

(8) 考试大：http://www.examda.com/ncre2/VB/zhenti/

(9) IT 认证考试网：http://www.itrz.cn/

(10) 中国 IT 考试论坛：http://bbs.cnitexam.com/

(11) 考试吧——全国计算机等级 NCRE 官方博客：http://ncre.blog.exam8.com/

(12) http://www.examda.com/soft/zhongji/pingce/

(13) http://www.exam8.com/computer/spks/rp/

17.4 四级软件测试工程师考试

17.4.1 概述

计算机技术与软件技术专业技术资格(水平)考试是一个难度很大的考试，考试内容涉及计算机专业的每门课程，还要加上数学、外语、系统工程、信息化和知识产权等知识，且注重考查新技术和新方法的应用。考试不但注重广度，而且还有一定的深度。从 2005 年上半年开始，计算机技术与软件技术专业技术资格(水平)考试中增加了软件评测师的考试，旨在培养软件评测师，为我国的软件评测提供专业人才。

全国计算机等级考试四级软件测试工程师(简称四级软件测试工程师)是全国计算机等级考试(四级)中的一类，属于计算机技术与软件专业资格(水平)考试的中级，每年只有上半年举办一次。考试分笔试和上机考试。笔试主要考查基础理论，上机考试考查实际工作能力。该考试主要考查软件测试的基本概念、结构，覆盖测试、功能测试、单元测试、集成测试、系统测试、软件性能测试、可靠性测试、面向对象软件测试、Web 应用软件测试以及兼容性测试、构件测试、极限测试和文档测试等。计算机四级软件测试工程师考试的合格考生应具有软件工程和软件质量保证的基础知识，掌握软件测试的基本理论、方法和技术，理解软件测试的规范和标准，熟悉软件测试过程，具备制定软件测试计划和大纲、设计测试用例、选择和运用测试工具、执行软件测试、分析和评估测试结果以及参与软件测试过程管理的能力，满足软件测试岗位的要求。

软件测试工程师考试的基本要求如下所示：

(1) 熟悉软件质量、软件测试及软件质量保证的基础知识；

(2) 掌握代码检查、走查与评审的基本方法和技术；

(3) 掌握白盒测试和黑盒测试的测试用例的设计原则和方法；

(4) 掌握单元测试和集成测试的基本策略和方法；

(5) 了解系统测试、性能测试和可靠性测试的基本概念和方法；

(6) 了解面向对象软件和 Web 应用软件测试的基本概念和方法；

(7) 掌握软件测试过程管理的基本知识和管理方法；

(8) 熟悉软件测试的标准和文档；

(9) 掌握 QESuite 软件测试过程管理平台和 QESat/C++软件分析和工具的使用方法。

1．题型分析

理论试卷的题型由选择题和填空题组成。选择题和填空题一般是对基本知识和基本操作进行考查的题型，它主要是测试考生对相关概念的掌握、理解是否准确，认识是否全面，

思路是否清晰，而很少涉及对理论知识的应用。上机考试主要考察考生对编程语言的掌握情况。具体地说，考试时应注意以下几个方面：

1) 选择题分析

选择题为单选题，是客观性试题，每道题的分值为 2 分，试题覆盖面广，一般情况下考生不可能做到对每个题目都有把握答对。这时，考生需要学会放弃，先易后难，对不确定的题目不要花费太多的时间，四级笔试题目众多，分值分散，考生一定要有全局观，合理地安排考试时间。

绝大多数选择题的设问是正确观点，称为正面试题；如果设问是错误观点，称为反面试题。考生在作答选择题时可以使用一些答题方法，以提高答题准确率。

(1) 正选法(顺选法)：如果对答案中的 4 个选项，一看就能肯定其中的 1 个是正确的，就可以直接得出答案。注意，必须要有百分之百的把握才行。

(2) 逆选法(排谬法)：逆选法是将错误答案排除的方法。对答案中的 4 个选项，知道其中的 1 个(或 2 个、3 个)是错误的，可以使用逆选法，即排除错误选项。

(3) 比较法(蒙猜法)：这种办法是没有办法的办法。

2) 填空题分析

填空填一般难度都比较大，需要考生准确地填入字符，往往需要非常精确，填错一个字也不得分。作答填空题时要注意以几点：

(1) 答案要写得简洁明了，尽量使用专业术语。

(2) 认真填写答案，字迹要工整、清楚，格式要正确，在把答案往答题卡上填写后尽量不要涂改。

(3) 向答题卡上填写答案时，一定要注意题目的序号，不要弄错位置。

(4) 对于那些有两种答案的填空题，只需填一种答案就可以了，多填并不多给分。

3) 上机试题分析

上机考试重点考察考生的基本操作能力和程序编写能力，要求考生具有综合运用基础知识进行实际操作的能力。上机试题综合性强、难度较大。上机考试的评分是以机评为主，人工复查为辅的。

上机考试应注意以下几点：

(1) 对于上机考试的复习，切不可“死记硬背”。考生一定要在熟记基本知识点的基础上，加强编程训练及上机训练，从历年试题中寻找解题技巧，理清解题思路，将各种程序结构反复练习。

(2) 一定要重视等级考试模拟软件的使用。在考试之前，应使用等级考试模拟软件进行实际的上机操作练习，尤其要做一些具有针对性的上机模拟题，以便熟悉考试题型，体验真实的上机环境。

(3) 学会并习惯使用帮助系统。每个编程软件都有较全面的帮助系统，熟练掌握帮助系统，可以使考生减少记忆量，解决答题过程中的疑难问题。

4) 理论考试综合应试分析

(1) 注意审题。命题人出题是有针对性的，考生在答题时也要有针对性。在解答之前，除了要弄清楚问题外，还有必要弄清楚命题人的意图，从而能够针对问题从容做答。

(2) 先分析，后下笔。明白了问题是什么以后，先把问题在脑海里过一遍，考虑好如何

做答后，再依思路从容做答。

(3) 对于十分了解或熟悉的问题，切忌粗心大意，而应认真分析，识破命题人设下的障眼法，针对问题，清清楚楚地写出答案。

(4) 对于不确定的题目，要静下心来，先弄清命题人的意图，再根据自己已掌握的知识的“蛛丝马迹”综合考虑，争取多拿一分是一分。

(5) 对于偶尔碰到的、以前没有见到过的问题或是虽然在复习中见过但已完全记不清的问题，也不要惊慌，关键是要树立信心，将自己的判断同书本知识联系起来做答。

(6) 对于完全陌生的问题，实在不知如何根据书本知识进行解答时，就可完全放弃书本知识，用自己的思考和逻辑推断作答。由于这里面有不少猜测的成分，能得几分尚不可知，故不可占用太多的时间。

(7) 理论考试时应遵循的大策略是：确保选择，力争填空。

总之，考试要取得好成绩，从根本上取决于考生对应试内容掌握的扎实程度。在比较扎实地掌握了应试内容的前提下，了解一些应试的技巧则能使考试成绩锦上添花。

2．考试方式

全国计算机等级考试四级软件测试工程师考试包括如下内容：

一、软件测试基本原理、测试方法、技术基础知识部分，采用笔试考试，考试时间 120 分钟，满分 100 分。

二、软件测试工程实践部分，上机操作完成下列内容：

(1) 软件测试过程管理实践，包括测试设计、测试准备、测试用例的执行、软件问题报告的填写、软件问题的跟踪解决。

内容描述：

① 给定一个被测系统的描述，要求建立测试项目组、分配人员角色、进行系统功能分解、编写测试用例。

② 执行测试，对于发现的测试问题填写软件问题报告。

③ 作为测试/开发人员，追踪处理问题报告的状态转换，直至问题的解决。

整个过程通过 QESuite 软件测试过程管理平台进行。

(2) 白盒测试实践。针对给定的被测程序设计测试用例进行测试，达到要求的语句覆盖率和分支覆盖率。

内容描述：

① 对于给定的 C 语言被测程序，编写测试用例。

② 使用 QESAT/C++白盒测试工具进行静态分析并插装被测程序。

③ 执行测试用例，进行动态测试。

④ 使用 QESAT/C++白盒测试工具检查测试覆盖率，直到达到所要求的覆盖率。

(3) 上机考试时间 120 分钟，满分 100 分。

3．应试技巧

针对考试大纲和考试要求进行复习，应注意以下几个方面：

(1) 牢固、清晰地掌握基本知识和理论。

“四级”考试的重点是实际应用和操作，但其前提条件是对基本知识点的掌握。那么，

考生正确地理解基本概念和原理便是通过考试的关键。应注意以下三点的复习：

① 在复习过程中要注意总结，只有通过综合比较、总结提炼才容易在脑海中留下清晰的印象和轮廓；

② 对一些重要概念的理解要准确，尤其是一些容易混淆的概念；

③ 通过联想记忆复习各考点，软件测试的知识点是相互联系的。

(2) 选择的习题要有针对性，切不可进行“题海战术”。

考生应根据考试大纲，在复习时适当地做一些与“四级”考试题型相同的题。研究过去、认识现在无疑是通过考试的一个重要的规律和诀窍，这么做可以使考生较快地熟悉考试题型，掌握答题技巧，从而能在最短的时间内收到最明显的效果，将往年习题进行适当分类整理，要通过做题掌握相关的知识点，真正做到“举一反三”。

(3) 复习笔试，上机实践。

复习笔试中有关程序设计的题目的最佳方法是上机操作，把程序在计算机上进行调试运行。

17.4.2　内容介绍

一、软件测试的基本概念

1. 软件质量的概念
2. 软件测试的目标和原则
3. 软件测试的心理学
4. 软件测试的经济学
5. 软件质量保证

二、软件测试的类型及其在软件开发过程中的地位

1. 软件开发阶段
2. 规划阶段的测试
3. 设计阶段的测试
4. 编码阶段的测试
5. 验收和维护阶段的测试

三、代码检查、走查与评审

1. 桌面检查
2. 代码走查
3. 代码检查
4. 同行评审

四、覆盖率(白盒)测试

1. 覆盖率测试
2. 逻辑结构的覆盖率测试
3. 路径覆盖率测试
4. 数据流测试

5. 程序变异测试
6. 基于覆盖的测试用例选择

五、功能(黑盒)测试

1. 边界值测试
2. 等价类测试
3. 基于因果图的测试
4. 基于决策表的测试
5. 基于状态图的测试
6. 基于场景的测试
7. 比较测试

六、单元测试和集成测试

1. 单元测试的目标和模型
2. 单元测试策略
3. 单元测试分析
4. 单元测试的测试用例设计原则
5. 集成测试基本概念
6. 集成测试策略
7. 集成测试分析
8. 集成测试用例设计原则

七、系统测试

1. 系统测试概念
2. 系统测试方法
3. 系统测试的实施

八、软件性能测试和可靠性测试

1. 软件性能的概念
2. 性能测试的执行
3. 软件可靠性的概念
4. 可靠性预计
5. 可靠性分析方法
6. 软件可靠性测试的执行

九、面向对象软件的测试

1. 面向对象软件测试的问题
2. 面向对象软件测试模型
3. 面向对象软件的测试策略
4. 面向对象软件的单元测试
5. 面向对象软件的集成测试
6. 面向对象软件的系统测试

十、Web 应用测试

1. 应用服务器的分类和特征
2. Web 应用系统的特点
3. Web 应用系统的测试策略
4. Web 应用系统测试技术
5. Web 应用系统安全测试

十一、其他测试

1. 兼容性测试
2. 易用性测试
3. GUI 测试
4. 构件测试
5. 极限测试
6. 文档测试

十二、软件测试过程和管理

1. 软件测试过程概念
2. 测试组织管理
3. 测试计划的制定
4. 测试步骤的确定
5. 测试环境管理
6. 软件测试风险分析和成本管理
7. 测试文档管理
8. 测试的复用与维护

十三、软件测试自动化

1. 测试自动化的原理、方法
2. 测试用例自动生成
3. 测试执行自动化
4. 测试结果比较自动化
5. 测试工具的分类和选择
6. 测试工具的主流产品介绍

十四、软件测试的标准和文档

1. 软件测试的标准
2. 软件测试的文档

十五、软件测试实践

1. 软件测试过程管理

(1) 软件测试过程管理概念。
(2) 测试的设计。
(3) 测试的准备。

(4) 测试的执行。

(5) 软件问题报告和软件问题生命周期。

(6) 测试的总结。

(7) QESuite 软件测试过程管理平台。

2. 白盒测试实践

(1) 被测程序说明。

(2) 静态分析。

(3) 被测程序的插装和动态测试。

(4) QESAT/C++白盒测试工具。

17.4.3 相关资料

[1] 张友生. 软件评测师考试考点分析与真题详解. 北京：电子工业出版社，2005

[2] 教育部考试中心. 全国计算机等级考试四级教程：软件测试工程师(2008 年版). 北京：高等教育出版社，2007

[3] 王健，苗勇，刘郢. 软件测试员培训教材. 北京：电子工业出版社，2003

[4] 蔡为东. 软件测试工程师面试指导. 北京：科学出版社，2007

第18章 测试行业

本章介绍当前软件测试行业。从测试行业现状、测试认识误区、测试工程师职位以及测试企业的面试题等方面进行了详细的说明。

18.1 测试行业概述

软件测试不仅早已成为软件开发的有机组成部分，而且在整个软件开发中占据着相当大的比重，所占比例高达 40%以上，同时占整个软件开发费用的 50%以上。软件测试人员与开发人员的人数比例大于 1∶2。例如，微软作为软件行业的龙头老大，开发 Windows 2000 操作系统共有 1700 多个开发人员，以及 3200 个测试人员，开发和测试人员之比约为 3∶5。HP 公司中测试人员和开发人员人数比例为 1∶1。

软件行业比较发达的国家，如欧美各国、印度、以色列等，软件测试行业的产值几乎占了软件行业总产值的 1/4。软件测试已经发展成为一个独立的产业，主要体现在：

(1) 软件测试在软件公司中占有重要地位，软件测试在人员配备和资金投入方面占据相当大的比重；

(2) 软件测试理论研究蓬勃发展，引领软件测试理论研究的国际潮流；

(3) 软件测试市场繁荣，如 MI、Compuware、Rational 等，其出品的测试工具占领了国际市场；

(4) 软件产品的认定往往需要第三方测试的介入。

随着中国软件业的日益壮大和逐步走向成熟，软件测试也在不断发展，从最初的由软件编程人员兼职测试到软件公司组建独立专职测试部门。测试工作也从简单测试演变为包括编制测试计划、编写测试用例、准备测试数据、编写测试脚本、实施测试、测试评估等多项内容的正规测试。测试方式则由单纯手工测试发展为手工、自动兼之，并有向第三方专业测试公司发展的趋势。

虽然我国软件产业以及信息化工作取得了较大成绩，但与发达国家相比，目前仍然停留在开发人员自行测试阶段，软件开发人员和测试人员结构明显失调，缺乏第三方测试。国内软件测试与国外软件测试主要存在如下差距：

(1) 测试的理解认识。国内软件企业普遍存在重开发轻测试，将测试置于从属地位，没有认识到软件项目完成不仅取决于开发人员，更取决于测试人员。国内许多中小型软件企业没有软件测试部门，有些甚至不设置软件测试的岗位。

(2) 测试过程的管理。国内软件企业普遍存在测试的随意化、简单化的情况，未建立有效、规范的测试管理体系。

(3) 测试工具的使用。国内软件企业的测试普遍缺乏使用自动化测试工具。

(4) 测试人员的培养。软件测试领域目前十分缺乏人才，首先是测试人员角色定位不合理，其次是缺乏专家级测试人才。

当然，国内软件测试产业也正在慢慢发展。第一，软件测试工具的开发取得了显著进展，如西安交通大学开发的 COBOL 测试系统、华中科技大学开发的 C 编译程序测试系统、北京航空航天大学与清华大学开发的 C 软件综合测试系统、中科院开发的 I-test 测试工具等。第二，软件企业对于软件测试设置了相应部门，如东软、神州数码等软件企业。第三，软件测试正在成为新的就业岗位。

18.2 测试认识误区

随着软件规模的扩大，软件设计复杂程度的提高，软件出现的缺陷越来越多，软件测试的重要性日益突出。但是，与软件开发相比，软件测试的地位和作用还没有真正受到重视，很多人(甚至是软件项目组的技术人员)对测试存在许多认识误区，影响测试活动的开展。下面介绍一些常见的测试认识的误区。

误区一：使用了测试工具，就是进行了有效的测试。

误区一是软件测试认识的通病，几乎是软件开发中每个领域 CASE 工具刚出现时都会产生的问题。比如：使用 Rational Rose 工具进行 UML 制图，就声称采用面向对象方法，而其软件设计和代码实现却是过程化的。同样，在测试中往往认为使用某种测试工具，就能获取种种好处，这种想法是错误的。

选择测试工具，应以是否满足需求为标准。对于大多数项目，一些开源测试工具的功能就足够了，比如，Java 语言的测试工具 JUnit 和 C++方面的单元测试工具 CppUnit 等。

误区二：软件中存在太多的无法测试的内容。

软件开发中确实存在一些内容很难测试，但并非无法测试。极限编程中，软件开发时首先编写测试代码，迫使开发者从易使用性、易测试性等角度考虑需求，专注软件模块的抽象，避免模块之间的耦合过紧，定义出清晰明确反映意图的模块接口，设计出良好的软件系统架构。

误区三：单元测试和验收测试没有什么区别。

以建筑行业类比软件开发过程，单元测试类比于建筑质检人员检测建筑时，关注的重点是建筑的内部结构、地基、框架以及墙壁是否垂直等，检测要保证建筑的各个部分符合建筑的质量标准。验收测试类比为建筑使用者对建筑进行的检测，使用者关注的重点是建筑的外观是否美观、各个房间的大小是否合适、窗户的位置是否合适、是否能够满足家庭的需要等。所以，建筑的使用者执行验收测试，是从用户的角度出发；而建筑质检人员执行单元测试，是从构建者的角度出发的。

单元测试和验收测试之间的区别取决于从不同的测试角度出发，二者互为补充。单元测试保证把事情做对，而验收测试则保证做正确的事情。关于单元测试和验收测试的划分，没有一个通用的标准。

下面给出一些指导原则：

(1) 单元测试要跨越类的边界，可能是一个验收测试。

(2) 单元测试变得非常复杂，可能是一个验收测试。

(3) 单元测试经常要随着用户需求的变化而改变，可能是一个验收测试。

(4) 单元测试比它要测试的代码本身要难以编写，可能是一个验收测试。

误区四：软件开发完成后进行软件测试。

一般认为，软件开发过程包括需求分析、概要设计、详细设计、软件编码、软件测试、软件发布等。据此，常常错误地认为软件测试是在软件编码之后进行的。

软件测试贯穿于软件项目的整个生命过程，在软件项目的每一个阶段都要进行不同目的和不同内容的测试活动，以保证各个阶段的正确性。软件测试的对象不仅仅是软件代码，还包括软件需求文档和设计等各类文档。

软件开发与软件测试是交互进行的。例如，编码需要单元测试，模块组合阶段需要集成测试。如果等到软件编码后才进行测试，测试的时间将会很短，测试的覆盖面将很不全面，测试的效果也将很差。

误区五：软件发布后发现质量问题，是测试人员的问题。

软件错误可能来自软件项目中的各个过程，软件测试只能确认软件存在错误，不能保证软件没有错误。从软件开发的角度看，高质量的软件不是软件测试的功劳，而依赖于软件生命周期的各个过程，特别是需求分析和设计阶段。如果软件质量出现问题，不能简单地归结为测试人员的责任，而应该分析软件过程的各个阶段，寻找出错的原因和改进的措施。

误区六：软件测试要求不高，随便找个人就行。

软件测试作为一个相对独立的领域，其技术不断更新和完善，新工具、新流程、新方法都在不断出现。随着软件测试行业的大力发展，需要更多具备测试技术和管理经验的测试人员。软件测试人员不仅需要掌握测试知识，更需要不断地实践。

误区七：项目进度吃紧时少做测试，时间宽裕时多做测试。

误区七是在软件开发过程中不重视软件测试的常见表现，也是软件项目过程管理混乱的表现。软件项目开发需要各种计划，对项目实施过程中的任何问题，都要有风险分析和相应的对策，不要因为开发进度的延期而简单地缩短测试时间、人力和资源。缩短测试时间带来测试的不完整，对项目质量的下降引入风险，会造成更大的软件缺陷。克服这种现象的最好办法是加强软件过程的计划和控制，包括软件测试计划、测试设计、测试执行、测试度量和测试控制。

误区八：软件测试是低级工作，开发人员才是软件高手。

开发和测试是相辅相成的过程，测试人员需要和程序员保持密切的联系，加强交流和协调。因此，软件测试不仅仅是测试人员的事情，也与程序员密切相关。软件测试人员和开发人员并没有技术的高低之分。软件开发人员往往只对自己开发的模块比较了解，对算法掌握的程度较高。而软件测试人员不仅要懂得如何测试，还要懂得被测软件的业务知识和专业知识。因此，软件测试和开发人员只是工作侧重点有所不同，并不存在水平差异的问题。

18.3 测试员的思维方式

微软亚洲工程院院长张宏江博士最近告诉媒体：“过去两三个月，我最主要的精力都花在雇人上。遗憾的是，1 万多名应聘者中，居然找不到足够合适的人。微软最紧缺的人才包

括软件测试人员、软件项目管理员、软件架构师，1 万多名应聘者中最后合格的只有 50 多人。”

软件测试需要学习的主要专业课程有：C 语言程序设计、Java 语言程序设计、软件工程与项目管理、数据库原理与应用、Linux 操作系统、网络应用技术、软件测试技术、软件测试过程管理、软件测试自动化、GUI 设计及测试、软件质量管理、IT 英语等。

作为软件测试人员，应具备以下几种思维方式：逆向思维方式、组合思维方式、全局思维方式、两极思维方式、简单思维方式和比较思维方式等

1．逆向思维方式

逆向思维是相对的，就是按照与常规思路相反的方向进行思考，比如根据结果逆推条件，从而得出输入条件的等价类划分。其实逆向思维在调试当中用到的也比较多，当发现缺陷时，进一步定位问题的所在，往往采用逆流而上进行分析。逆向思维往往能够发现开发人员思维的漏洞。

2．组合思维方式

很多东西单一的思考都没有问题，当将相关的事物组合在一起却能发现很多问题，如多进程并发，不但使得程序的复杂度提高，也让程序的缺陷率随之增长。针对不同的应用，可以酌情考虑使用“排列”或者“组合”，将相关的因素划分到不同的维度上，然后再考虑其相关性。

3．全局思维方式

全局思维方式就是从多角度分析待测的系统，以不同角色去看系统，分析其是否能够满足需求。在软件开发过程中进行的各种评审，应让更多的人参与思考，尽可能地实现全方位审查某个解决方案的正确性以及其它特性。

4．两极思维方式

两极思维方式，是在极端的情况下，看是否存在缺陷。边界值分析方法就是两极思维方式的典范。

5．简单思维方式

通过舍去一些非关键特征，针对事物本质进行测试，这样不至于偏离方向。

6．比较思维方式

人们认识事物时，往往都是和已有的某些概念进行比较，找出相同、相异之处，或者归类。应用模式是“比较思维”很常见的例子，有设计模式、体系结构模式、测试模式等。由于经验在测试中很重要，因此比较思维是较为常用的方式。

18.4 著名企业的测试面试题

1. 软件测试分哪两种方法？分别适合什么情况？

【解析】 软件测试方法一般分为白盒测试与黑盒测试。白盒测试又称为结构测试、逻辑驱动测试或基于程序本身的测试，它着重于程序的内部结构及算法，通常不关心功能与性能指标；黑盒测试又称为功能测试、数据驱动测试或基于规格说明的测试，它实际上是

站在最终用户的立场，检验输入输出信息及系统性能指标是否符合规格说明书中有关功能需求及性能需求的规定。

2. 一套完整的测试应该由哪些阶段组成？分别阐述各个阶段。

【解析】 完整的测试由计划阶段、设计阶段、白盒单元、白盒集成、黑盒单元、黑盒集成、系统测试、回归测试、验收测试等一套完整的测试组成，一般包括五个阶段：

(1) 测试计划。首先，根据用户需求报告中关于功能要求和性能指标的规格说明书，定义相应的测试需求报告，即制订黑盒测试的最高标准。以后所有的测试工作都将围绕着测试需求来进行，符合测试需求的应用程序即是合格的，反之即是不合格的；同时，还要适当选择测试内容，合理安排测试人员、测试时间及测试资源等。

(2) 测试设计将测试计划阶段制订的测试需求分解、细化为若干个可执行的测试过程，并为每个测试过程选择适当的测试用例(测试用例选择的好坏将直接影响测试结果的有效性)。

(3) 测试开发建立可重复使用的自动测试过程。

(4) 执行测试开发阶段建立的自动测试过程，并对所发现的缺陷进行跟踪管理。测试执行一般由单元测试、组合测试、集成测试、系统联调及回归测试等步骤组成，测试人员应本着科学负责的态度，一步一个脚印地进行测试。

(5) 测试评估，结合量化的测试覆盖域及缺陷跟踪报告，对于应用软件的质量和开发团队的工作进度及工作效率进行综合评价。

3. 您是否了解以往所工作的企业的软件测试过程？如果了解，请试述在这个过程中都有哪些工作要做？分别由哪些不同的角色来完成这些工作？

【解析】 软件测试部门配合系统分析人员对软件需求进行分析讨论，并根据需求说明书制定项目测试计划，编写测试用例，建立测试环境。

软件测试人员负责软件开发部门的新产品测试及原有产品的升级测试，负责软件问题解决过程跟踪，负责软件开发文档开发工作的规范化及管理开发部门的产品文档，制作用户手册及操作手册，负责产品的上线测试，监督软件开发过程的执行，提高产品质量。

4. 解释验证测试、基于用户实际应用场景的测试、冒烟测试、兼容性测试及适用性测试的区别与联系。

【解析】 验证测试的主要目的是验证最新生成的软件版本在功能上是否完整，主要的软件特性是否正确；基于用户实际应用场景的测试优点是关注了用户的需求，缺点是有时候难以真正模仿用户真实的使用情况；冒烟测试指修复 Bug 后，针对此次修复是否会对其它模块造成影响而进行的专门测试，其优点是节省测试时间，防止建构失败，缺点是覆盖率还是比较低；兼容性测试，主要目的是为了兼容第三方软件，确保第三方软件能正常运行，用户不受影响；适用性测试，是确保软件对于某些有残疾的人士也能正常地使用，但优先级比较低。其它的测试还有功能测试、安全性测试、压力测试、性能测试、回归测试、安装升级测试等。

5. 测试用例通常包括哪些内容？着重阐述编制测试用例的具体做法的不同结构(版本、编号、项目、设计人员、设计日期、输入、预期输出等)。

【解析】 软件测试用例的基本要素包括测试用例编号、测试标题、重要级别、测试输入、操作步骤、预期结果。

用例编号：测试用例的编号有一定的规则，比如系统测试用例的编号这样定义规则：PROJECT1-ST-001，命名规则是项目名称+测试阶段类型(系统测试阶段)+编号。定义测试用例编号，便于查找测试用例，便于测试用例的跟踪。

测试标题：对测试用例的描述。测试用例标题应该清楚表达测试用例的用途。比如“测试用户登录时输入错误密码时，软件的响应情况”。

重要级别：定义测试用例的优先级别，可以笼统地分为“高”和“低”两个级别。一般来说，如果软件需求的优先级为“高”，那么针对该需求的测试用例优先级也为“高”；反之亦然。

测试输入：提供测试执行中的各种输入条件。根据需求中的输入条件，确定测试用例的输入。测试用例的输入对软件需求当中的输入有很大的依赖性，如果软件需求中没有很好地定义需求的输入，那么测试用例设计中会遇到很大的障碍。

操作步骤：提供测试执行过程的步骤。对于复杂的测试用例，测试用例的输入需要分几个步骤完成，这部分内容在操作步骤中详细列出。

预期结果：提供测试执行的预期结果，预期结果应该根据软件需求中的输出得出。如果在实际测试过程中，得到的实际测试结果与预期结果不符，那么测试不通过；反之则测试通过。

6. 描述使用 Bugzilla 缺陷管理工具对软件缺陷(Bug)跟踪管理的流程。

【解析】 测试人员或开发人员发现 Bug 后，首先判断属于哪个模块的问题，填写 Bug 报告后，系统会自动通过 E-mail 通知项目组长或直接通知开发者。

(1) 经验证无误后，修改状态为“核实”。待整个产品发布后，修改为关闭，如果还有问题，重新打开，状态重新变为“New”，并发邮件通知。

(2) 项目组长根据具体情况，重新再次分配给 Bug 所属的开发者。

(3) 开发者收到 E-mail 信息后，判断是否为自己的修改范围。

(4) 若是，进行处理并给出解决方法；若不是，重新再次分配给项目组长或应该分配的开发者。

(5) 测试人员查询开发者已修改的 Bug，进行重新测试。

7. 您所熟悉的测试用例设计方法都有哪些？请分别以具体的例子来说明这些方法在测试用例设计工作中的应用。

【解析】 常用的测试用例设计方法分为黑盒和白盒两种测试。黑盒测试有等价类划分法、边界分析法、因果图法和错误猜测法；白盒测试有逻辑覆盖法、循环测试路径选择、基本路径测试。例子：在一次输入多个条件的完整性查询中，利用等价类划分法则和边界分析法则。首先利用等价类划分法，可以划分出一个或多个结果是“OK”的测试用例，多个结果是“NG”的测试用例，然后利用边界值分析法，对结果分别是“OK”和“NG”的测试用例进行扩展和补充。

8. 您认为做好测试用例设计工作的关键是什么？

【解析】 测试用例设计工作的关键是对可行的和不可行的情况都要予以考虑。具体包括：① 输入；② 详细的操作步骤；③ 预期输出；④ 实际输出。

9. 您在从事性能测试工作时，是否使用过一些测试工具？如果有，请试述该工具的工作原理，并以一个具体的工作中的例子描述该工具是如何在实际工作中应用的。

【解析】 使用 LoadRunner 测试工具。该工具能够录制测试人员的操作步骤，然后对这个操作步骤模拟出多个用户。具体步骤如下：

(1) Virtual User Generator 创建脚本，选择协议，录制操作，编辑操作。

(2) 中央控制器(Controller)调度虚拟用户，创建场景，选择脚本，建立虚拟用户，设计shedual，设置 ip spoofer。

(3) 运行脚本，分析 shedual。

(4) 分析测试结果。

10. 您认为性能测试工作的目的是什么？做好性能测试工作的关键是什么？

【解析】 性能测试工作的目的是检查系统是否满足需求说明书中规定的性能，性能测试常常需要和强度测试结合起来，要求同时进行软件和硬件的检测。性能测试主要的关注对象是响应时间、吞吐量、占用内存大小(辅助存储区)、处理精度等。

11. 在您以往的工作中，一条软件缺陷(Bug)记录都包含了哪些内容？如何提交高质量的软件缺陷记录？

【解析】 一条软件缺陷记录所包括的内容有：检测时间、系统环境、硬体环境、严重程度、程式版本、确认人、功能模块、问题描述、详细操作步骤及是否会重现等。问题描述和详细操作步骤要尽可能的详细。Bug 应该尽量用书面语，对于严重程度比较高的缺陷要在相同环境下测试。在 C/S 模式下，如果条件满足可以使用替换法来确认是 Client 端的问题还是 Server 端的问题。

12. 你在测试中发现了一个 Bug，但是开发经理认为这不是一个 Bug，你应该怎样解决？

【解析】 具体解决步骤如下：

(1) 将问题提交到缺陷管理库进行备案。

(2) 获取 Bug 判断的依据和标准：

① 根据需求说明书、产品说明、设计文档等，确认实际结果是否与计划有不一致的地方，提供缺陷是否确认的直接依据；

② 如果没有文档依据，可以根据类似软件的一般特性来说明是否存在不一致的地方，以确认是否是缺陷；

③ 根据用户的一般使用习惯，确认是否存在缺陷；

④ 与设计人员、开发人员和客户代表等相关人员探讨，确认缺陷。

(3) 合理地论述，向测试经理说明自己的判断理由，注意客观、严谨、不掺杂个人情绪。

(4) 等待测试经理做出最终决定，如果仍然存在争议，可以通过公司政策所提供的渠道，向上级反映，并由上级做出决定。

18.5 软件测试工程师职位简介

下面是 IT 行业中软件测试工程师职位的发展之路。

(1) 初级测试工程师。刚入门拥有计算机科学学位的个人或具有一些手工测试经验的个人。开发测试脚本并开始熟悉测试生存周期和测试技术。

(2) 测试工程师/程序分析员。具有 1～2 年工作经验的测试工程师或程序员。编写自动

测试脚本程序并担任测试编程初期领导工作。拓展编程语言、操作系统、网络与数据库技能。

(3) 高级测试工程师/程序分析员。具有3～4年工作经验的测试工程师或程序员。帮助开发或维护测试或编程标准与过程，负责同级的评审，并为其它初级的测试工程师或程序员充当顾问。

(4) 测试组负责人。具有4～6年工作经验的测试工程师或程序员。负责管理1～3名测试工程师或程序员。担负一些进度安排和工作规模/成本估算职责。

(5) 测试/编程负责人。具有6～10年工作经验的测试工程师或程序员。负责管理8～10名技术人员。负责进度安排、工作规模/成本估算，按进度表和预算目标交付产品。

(6) 测试/质量保证/开发(项目)经理。具有10多年的工作经验。管理8名或更多的人员参加1个或多个项目。负责这一领域(测试/质量保证/开发)内的整个开发生存周期业务。

(7) 计划经理。具有15年以上开发与支持(测试/质量保证)活动方面的工作经验。管理从事若干项目的人员以及整个开发生存周期。负责把握项目方向并担负盈亏责任。

参 考 文 献

[1] Chris Wysopal，Lucas Nelson，Dino Dai Zovi，et al. 软件安全测试艺术. 程永敬，译.北京：机械工业出版社，2007.

[2] Paul C.Jorgensen. 软件测试. 2 版. 韩柯，杜旭涛，译.北京：机械工业出版社，2007.

[3] Srinivasan Desikan，Gopalaswamy Ramesh. 软件测试原理与实践. 韩柯，李娜，译. 北京：机械工业出版社，2007.

[4] 马瑟. 软件测试基础教材(英文版). 北京：机械工业出版社，2008.

[5] Daniel J.Mosley，Bruce A.Posey. 软件测试自动化. 邓波，黄丽娟，曹青春，等，译. 北京：机械工业出版社，2003.

[6] 麦格雷戈，等. 面向对象的软件测试. 杨文宏，等，译. 北京：机械工业出版社，2002.

[7] Glenford J.Myers，et al. 软件测试的艺术. 2 版. 王峰，陈杰，译. 北京：机械工业出版社，2006.

[8] Dirk Huberty，et al. 软件质量和软件测试. 马博，赵云龙，译. 北京：清华大学出版社，2003.

[9] David Gustafson. 软件工程习题与解答. 钟鸣，王君，等，译. 北京：机械工业出版社，2003.

[10] 段念. 软件性能测试过程详解与案例剖析. 北京：清华大学出版社，2006.

[11] 赵池龙. 实用软件工程. 北京：电子工业出版社，2003.

[12] 史九林，古乐. 软件测试技术概论. 北京：清华大学出版社，2004.

[13] 董杰. 软件测试精要. 北京：电子工业出版社，2008.

[14] 贺平. 软件测试教程. 北京：电子工业出版社，2006.

[15] 宏刚，等. 软件开发的科学与艺术. 北京：电子工业出版社，2002.

[16] 宫云战. 软件测试教程. 北京：机械工业出版社，2008.

[17] 路晓丽，葛玮，龚晓庆，等. 软件策划斯技术. 北京：机械工业出版社，2007.

[18] 宫云战. 软件测试. 北京：国防工业出版社，2006.

[19] 王东刚. 软件测试与 Junit 实践. 北京：人民邮电出版社，2004.

[20] 王健，苗勇，刘郢. 软件测试员培训教材. 北京：电子工业出版社，2003.

[21] 朱少民. 软件测试方法和技术. 北京：清华大学出版社，2005.

[22] 李幸超. 实用软件测试——来自硅谷的技术、经验、心得和实例. 北京：电子工业出版社，2006.

[23] 张友生. 软件评测师考试考点分析与真题详解. 北京：电子工业出版社，2005.

[24] 郑人杰. 计算机软件测试技术. 北京：清华大学出版社，1992.

[25] 魏方. 计算机软件测试技术与实例. 北京：北京希望电脑公司，1991.

[26] 麦格雷戈， McGregor Sykes，杨文宏. 面向对象的软件测试. 北京：中信出版社，2002.

[27] 朱少民. 全程软件测试. 北京：电子工业出版社，2007.

[28] 教育部考试中心. 全国计算机等级考试四级教程：软件测试工程师(2008 年版). 北京：高等教育出版社，2007.

[29] Patton，周予滨，姚静. 软件测试. 北京：机械工业出版社，2002.

[30] 秦晓. 软件测试. 北京：科学出版社，2008.

[31] 佟伟光. 软件测试. 北京：人民邮电出版社，2008.

[32] Gerald D.Everett，Raymond Mcleod，Jr，郭耀. 软件测试：跨越整个软件开发生命周期. 北京：清华

大学出版社，2008.

[33] 古乐，史九林，等. 软件测试案例与实践教程. 北京：清华大学出版社，2007.

[34] 蔡为东. 软件测试工程师面试指导. 北京：科学出版社，2007.

[35] 陈能技. 软件测试技术大全=Software Testing GuideFundamentals，Tools and Practice: 测试基础 流行工具 项目实战. 北京：人民邮电出版社，2008.

[36] 陈文滨，朱小梅，任冬梅. 软件测试技术基础. 北京：清华大学出版社，2008.

[37] Mark Fewster & Dorothy Graham，舒智勇. 软件测试自动化技术与实例详解. 北京：电子工业出版社，2000.

[38] 张克东，庄燕滨. 软件工程与软件测试自动化教程. 北京：电子工业出版社，2002.

[39] 林宁，孟庆余. 中国电子技术标准化研究所. 软件测试实用指南. 北京：清华大学出版社，2004.

[40] 刘怀亮，相洪贵. 软件质量保证与测试. 北京：冶金工业出版社，2007.

[41] 周伟明. 软件测试实践. 北京：电子工业出版社，2008.

[42] 许育诚. 软件测试与质量管理. 北京：电子工业出版社，2004.

[43] 赵斌. 软件测试技术经典教程. 北京：科学出版社，2007.

[44] 韩万江. 软件工程案例教程. 北京：机械工业出版社，2009.

[45] 郁莲. 软件测试方法与实践. 北京：清华大学出版社，2008.

[46] 张向宏. 陈渌萍. 工业和信息化部电子教育与考试中心组. 软件测试理论与实践教程. 北京：人民邮电出版社，2009.

[47] 周元哲. 软件测试教程. 北京：机械工业出版社，2010.

[48] http://www.uml.org.cn/Test/test.asp
软件测试 软件测试方法 软件测试工具 软件测试工程师 功能测试 性能测试 自动化测试 测试用例 测试管理-UML 软件工程组织

[49] http://www.51testing.com/html/index.html
51Testing 软件测试网-中国软件测试人的精神家园

[50] http://www.cntesting.com/
软件测试基地

[51] http://www.17testing.com/
一起测试网 软件测试专家 测试培训 软件测试 软件测试技术 测试 一起测试 test 17testing 测试工具 测试报告 自动化测试 软件测试培训 单元测试 系统测试 软件测试工具 软件测试咨询 软件测试外包 QTP LR

[52] http://testing.csdn.net/
CSDN--软件测试频道

[53] http://www.exam8.com/computer/spks/rp/
软件评测师-软件水平考试-考试吧

[54] http://www.examda.com/soft/zhongji/pingce/
全国计算机软考软件评测师考试-软件评测师考试大纲-软件评测师习题-软件评测师辅导资料-计算机软件水平考试-考试大

[55] http://www.iturls.com
IT 之源

[56] http://www.spasvo.com/
泽众软件-软件测试工具|系统自动化测试软件|软件测试学习与入门